

# Bachelor's thesis

## TuberXpert



# TUCUXI

Il faut présenter un peu ce qu'est un drug model avant de détailler les specs, sinon on perd le lecteur.

**Student:**

**Melvyn Herzig**

**Work proposed by:**

Professor Yann Thoma

REDS

Route de Chesaux 1

1401 Yverdon-les-Bains

**Responsible teacher:**

**Professor Yann Thoma**

**Academic year:**

2021-2022

Yverdon-les-Bains on Monday, May 16, 2022

# TuberXpert specifications

## Context

This work takes place in the domain of therapeutic drug monitoring (TDM). Currently, Tanzania is experiencing a high amount of tuberculosis (TB) cases. The country has expressed its commitment to end TB by 2035. The main tool to achieve this goal is TDM. However, the problem is that it is a long and difficult process. A software called Tucuxi has already been developed and is part of the solution. Nevertheless, it does not solve the fact that the tool needs TDM professionals to be used efficiently. On this perspective, the purpose of this work is to develop a clinical decision support system (CDSS) on top of Tucuxi computation core to automate TDM decisions making.

## Objective

By the 29<sup>th</sup> of July, an extensible CDSS must be developed, tested and documented. The CDSS will use a local version of Tucuxi computation core for dosage prediction and adjustment computations. The system will be a command line interface that will produce a dosage adjustment report based on the received inputs.

## Features

### Input validation

- The program will receive an XML file similar to Tucuxi computation core.
  - The XML structure may be extended with new useful elements if needed.
- The program will analyze and check data relevance.

### Drug file selection

- The program must be able to select a relevant drug file for each drug in input.

### Dosage adjustment

- The program must be able to understand the current state of a treatment and suggest an adjustment.

### Output

- The output of the decisions must be an XML file that can be used by various templates for the report generation.

### Report generation

- The program must summarize all useful information in a well-formatted report.
  - Suspicious covariates.
  - Drug file selected.
  - Graph “A priori” or “A posteriori”, depending on the patient.
  - Dosage adjustment.
  - ...

### Multi-language

- The program must support various languages.
- At least, the English version must be available.
  - It should be easy to add a translation and use it.

### Testing

- The program behavior must be tested with various inputs.
  - Since it is difficult to predict all cases, obvious cases testing is sufficient.

## Deadlines

When	What
16.05.2022	Intermediate report.
29.07.2022 12:00	Upload final report, poster and publishable summary on GAPS.
29.07.2022 12:00	Notify responsible and secretariat by email when previous point is made.
22.08.2022 – 16.09.2022	Bachelor thesis defense.

# Table of contents

1	Preamble.....	7
2	Introduction .....	8
2.1	What is therapeutic drug monitoring.....	8
2.2	Current situation in Tanzania .....	8
2.3	Goal of this work.....	9
3	Software needs .....	10
3.1	Requirements .....	10
3.1.1	Functional .....	10
3.1.2	Nonfunctional .....	10
3.2	Testing .....	10
3.3	Architecture .....	10
4	Analysis .....	11
4.1	Global application overview .....	11
4.2	Technologies.....	11
4.2.1	Development language.....	11
4.2.2	Integrated Development Environment .....	11
4.2.3	Compiler .....	12
4.3	Input.....	12
4.3.1	Global.....	12
4.3.2	Covariates .....	13
4.3.3	Drugs.....	13
4.3.4	Dosage .....	14
4.3.5	Samples.....	15
4.3.6	Target.....	15
4.3.7	Request.....	16
4.4	Program execution flow .....	17
4.4.1	Get best drug file .....	18
4.4.2	Assess the dosages .....	19
4.4.3	Assess the samples .....	20
4.4.4	Assess the targets .....	21
4.4.5	Create adjustment request.....	22
4.5	Output.....	24
4.5.1	Header .....	24
4.5.2	Administrative .....	25
4.5.3	Covariates and checks .....	26

4.5.4	Treatment and checks .....	27
4.5.5	Samples and check.....	28
4.5.6	Adjustments.....	29
4.5.7	Computation facts .....	32
5	Implementation .....	34
5.1	Query importer.....	34
5.1.1	XML query structure .....	34
5.1.2	Administrative data .....	40
5.1.3	RequestXpert data .....	41
5.1.4	Query data .....	41
5.1.5	Query importation .....	42
5.2	Language manager .....	42
5.2.1	XML dictionary structure .....	42
5.2.2	Getting a translation.....	43
6	Tests.....	44
6.1	Unit Tests .....	44
6.1.1	LanguageManager .....	44
6.1.2	XpertQueryImport .....	44
7	Conclusion.....	48
8	Bibliography .....	49
9	Authentication .....	50
10	List of abbreviations.....	51
11	List of figures.....	52
12	Planification .....	53
12.1	Initial .....	53
12.2	Intermediate.....	54
12.3	Final .....	55

Bachelor's thesis 2021-2022

TuberXpert

---

**Publishable summary**

<Coming soon>

Student:

Herzig Melvyn

Responsible teacher:

Thoma Yann

Date and time:

.....

Date and time:

.....

Signature:

.....

Signature:

.....

# **1 Preamble**

This Bachelor's thesis is produced at the end of the course of study, with a view to obtaining the title of Bachelor of Science HES-SO in Engineering.

As an academic work, its content, without prejudging its value, does not engage the responsibility of the author, nor that of the jury of the Bachelor's thesis and of the school.

Any use, even partial, of this Bachelor's thesis must be made in compliance with the copyright.

HEIG-VD

The head of the department

Yverdon-les-Bains on Friday, March 4, 2022

## 2 Introduction

### 2.1 What is therapeutic drug monitoring

Nowadays, many drugs or antibiotics are used to treat diseases such as tuberculosis and HIV. Usually, the doctors prescribe generic doses that are suitable for the general population. Unfortunately, everyone's metabolism reacts differently which makes generic dosages often ineffective.

Some people will have insufficient circulating drug exposure caused by an underdose. Thus, the treatment will be ineffective, and the patient may become drug resistant. Conversely, an overdose may result in intoxication. This would force an interruption of the treatment in order not to worsen the patient's health.

To avoid such situations, therapeutic drug monitoring has been developed. TDM is a precision medicine that prescribes a personalized dosage to each patient based on the monitoring of the evolution of the drug concentration in the blood.

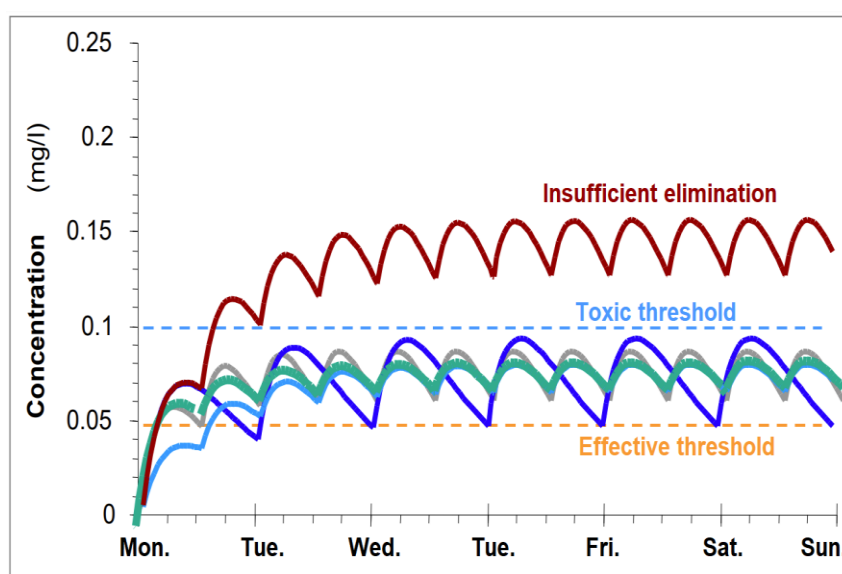


Figure 1 - Dosage Scheme (BUCLIN Thierry, 2022, *Les Bases de la pharmacocinétique clinique* [PDF document])

The purpose of TDM is to measure the concentration of the drug regularly and accurately in order to know its evolution. After examination, an expert in pharmacology can determine how to adjust the dosage to be above the **threshold of inefficiency** and below the **threshold of toxicity**.

### 2.2 Current situation in Tanzania

Tanzania has a high burden of tuberculosis. Over the last decade, Tanzanian health authorities estimate an incidence of 120'000 – 150'000 patients per year for TB. The global community, through the END TB strategy, has declared its willingness to end TB by 2035 and a vital component of the arsenal for this includes resorting to the correct use of anti-TB drugs, in particular the first-line agents: isoniazid, rifampicin, ethambutol and pyrazinamide.

However, the problem is that only 60'000 – 75'000 patients are notified and receive a treatment. In addition, studies have reported that rifampicin dosages are insufficient most of the time. For example, investigations by Heysell et al. (2011) and Tostmann et al. (2013) in the Kilimanjaro region showed that one to two thirds of uncomplicated TB patients had maximum concentrations below the reference range of 8-24 mg/L, defined two hours after the last dose intake.

Moreover, a considerable proportion of individuals with TB are co-infected with HIV. It represents 25 to 40% of the monitored people. Administration of antiretrovirals with first-line anti-tuberculosis drugs further reduces the concentration of rifampicin.



On top of that, TB patients have an increased risk to get affected by diabetes mellitus (DM). It represents 4-16% of the TB population. Unluckily, DM may alter the pharmacokinetics (PK) of various drugs which include antitubercular. Mtabho et al. observed that DM predicted low levels of rifampicin in TB Tanzanian patients. Sadly, evidence has shown that individuals with TB and DM are five times more likely to die than those without diabetes.

At this point, we easily understand that the risk of treatment failure or unfavorable outcome is real if the dosages stay unsuitable.

## 2.3 Goal of this work

The need to end tuberculosis is real and urgent in Tanzania. Unfortunately, TDM is a long and complicated process. In addition, the number of experienced pharmacologists is not sufficient to provide well-established interpretation and recommendation everywhere their expertise is needed.

Currently, Professor Yann Thoma and his team have developed Tucuxi. It is a software intended for the practice of TDM. Already developed for several years, the software offers many features:

- Drug concentration predictions based on population and patient data (covariates) as well as on previous measurements.
- Suggestion of dosage adjustments to reach an optimal drug concentration state.
- Generation of printable reports.
- Integration with Electronics Health Record (EHR) systems.

Although it simplifies the TDM process, Tucuxi is intended for experienced pharmacologists.

TuberXpert comes at the beginning of a large project between Switzerland and Tanzania led by Prof. Thoma Yann, Prof. Mpagama Stellah and Prof. Guidi Monia. The aim is to develop a Clinical Decision Support System to fight tuberculosis. Thus, TuberXpert is a software layer that adds to the existing Tucuxi computing core. By receiving complete information from a patient, the system assesses the relevance of the data provided and then determines whether an adjustment of dosage is necessary. All interpretations made by the program will then be provided to the user in the form of a simplified report compared to the original software. The main purpose of TuberXpert is to simplify the “interpretation and recommendation” phase of TDM for non-experts.

In other words, TuberXpert is a turnkey solution for TDM. The software developed during this Bachelor's thesis is a first step. It will then be taken over by three Ph.D. students in charge of the development and the concrete application of the project.

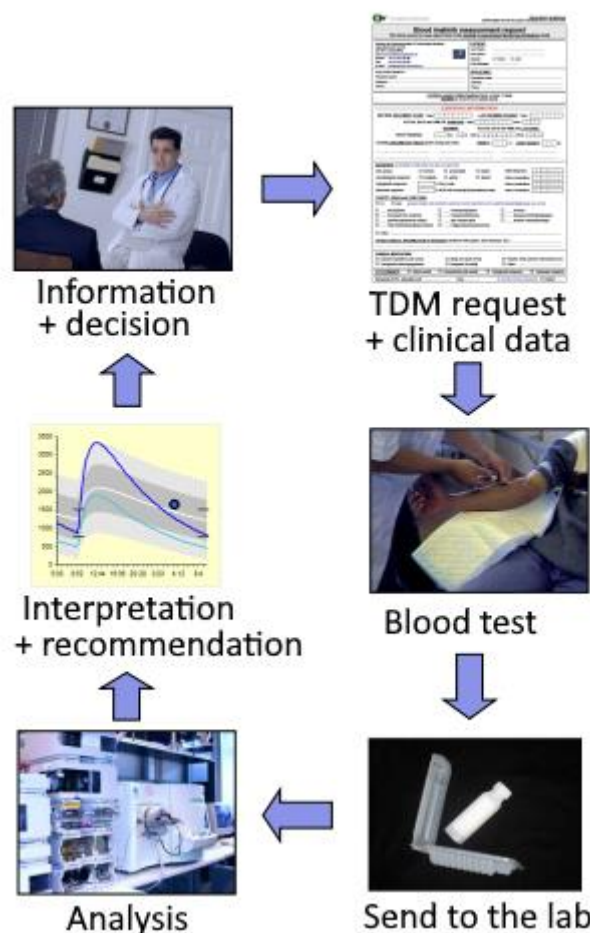


Figure 2 -TDM process (BUCLIN Thierry, 2022, Les Bases de la pharmacocinétique clinique [PDF document])

## 3 Software needs

This chapter presents a specification of what is to be implemented. It describes how the system should behave and what its properties are.

### 3.1 Requirements

#### 3.1.1 Functional

**From the user's point of view:** TuberXpert is an automated clinical decision support system.

It takes as input a query: some information about a patient and a request. Then, thanks to an output report, the program suggests an initial dosage or an adjustment of the drugs and provides any useful information for the therapeutic drug monitoring. The output contains readable sentences, and the report displays some graphs of the adjustment. **Also warnings about issues**

**From the developer's point of view:** TuberXpert is an additional software layer on top of Tucuxi computing core.

As input, TuberXpert needs an XML file containing the patient's information and the request to execute. The structure of the file is the same as that of Tucuxi computing core. However, it could extend the basic structure by adding some new elements, at least, a new type of request dedicated to TuberXpert control.

The first step in the execution is to obtain a relevant drug file. TuberXpert does not ask the user to specify which drug model to use. It chooses the most appropriate one based on the available patient information. **There are several models for a drug?**

Then, the program performs a validity check on the data provided. It assesses whether the values received are plausible. Are they normal? Did the practitioner make a typing error?

After the validation step, TuberXpert prepares an adjustment request that it submits to Tucuxi computing core.

Finally, it generates an output report corresponding to a format requested by the user. This can be XML, HTML or PDF. The report contains full sentences and the adjustment graphs for the HTML and PDF format or the adjustment data to generate a graph for the XML format. The report can be generated in several languages.

#### 3.1.2 Nonfunctional

This project is most likely a proof of concept. For this work, it is not necessary to develop a graphical user interface. The program will be run in a command-line interface.

As long as it is relevant, the generated report must be good-looking and user-friendly. In other words, it should not be painful to read and to extract the information.

Although this work is made in the context of tuberculosis, it is developed without an extensive knowledge of rifampicin or any other drug. Consequently, the development is generic in considering all drugs. Therefore, it should be easy to edit the parts specific to the adaptation of a drug.

### 3.2 Testing

The program behavior must be evaluated with various inputs. Since it is difficult to predict all cases, obvious cases testing is enough.

### 3.3 Architecture

In terms of software architecture, the clinical decision support system may be separated from the report generation. The CDSS must offer a standardized output, most probably in XML format. This output may be used by a third-party report generator to fill some fields of a report template.

## 4 Analysis

This chapter presents the analysis that is done prior to the implementation. The objective is to anticipate and take some decisions based on business and technical analysis.

### 4.1 Global application overview

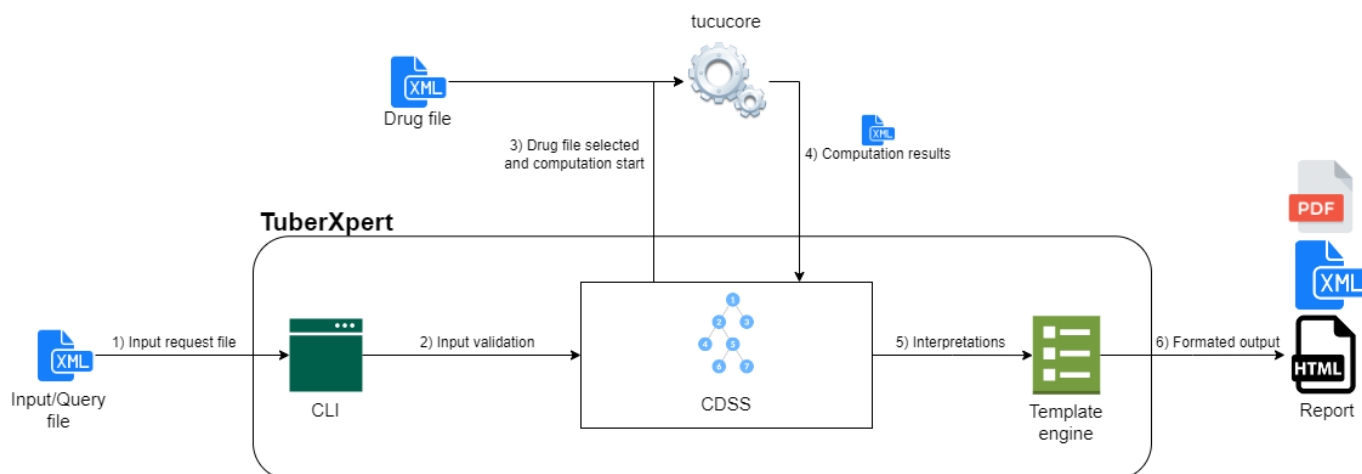


Figure 3 Global application overview and components

1. The program receives as input a query in XML format. For simplification purpose, the XML format will be the same used by Tucuxi computing core. However, it will be extended with a specific request and administrative elements.
2. The program loads the data from the query file and transmit them to the CDSS that determines if they are consistent and relevant.
3. The CDSS determines the best drug model to be used and starts the computation.
4. The computation results are transmitted as XML to the CDSS which analyzes them.
5. The decisions are passed to a template engine that formats the information into the requested format.
6. The report is returned.

## 4.2 Technologies

### 4.2.1 Development language

Initially, Tucuxi was developed in C++. This language was chosen for its superior performance since the software requires a lot of computing performance. Then, other projects were added. Most of them were also developed in C++ for the same reasons.

Thus, to preserve the homogeneity of the projects, TuberXpert will also be developed in C++. This language is once again very advantageous. As the software could be used on low-powered computers in Tanzania, the performance will be optimized by using a low-level language.

Version: C++ 17

### 4.2.2 Integrated Development Environment

QT is not QtCreator

Once again, it will walk in the steps of Tucuxi, and it will use the same development environment. QT is a very advantageous and rich IDE. It easily allows cross-platform development, test management, language management and many other frameworks. QT is very versatile. It also makes it easier to work with existing projects at the same time. De facto, it is the IDE that will be used to develop this project.

Version: QT 6.0.2 community

### 4.2.3 Compiler

To simplify the development environment installation, this project is compiled with the compiler integrated with QT.

Version (Windows): MinGW-W64-builds-5.0.0

## 4.3 Input

The TuberXpert query XML file will be responsible for providing all the useful information about the patient, his treatments and the computation to be performed. Most of its structure will be essentially the same as Tucuxi computing core<sup>1</sup>.

Although, this section goes through the input structure, it does not explain how to form the common elements with Tucuxi computing core but what they represent and, if necessary, what will be checked.

When a pertinent spot<sup>1</sup> is reached, it explains what additional element needs to be added to fit TuberXpert needs such as administrative data and the custom request.

### 4.3.1 Global

Down below is the overall structure of the input. It consists of information about the query: **queryId**, **clientId**, **date** and **language**. Then, there is information about the patient's **covariates** followed by the **drugs** he is taking. Finally, comes the **requests** element. It contains **request** elements used to tell Tucuxi computing core what calculation to perform from what data, but it is ignored by TuberXpert since it will create these requests itself. However, it will include a new type which is the TuberXpert request: **requestXpert**.

Example of the global structure of the input:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<query version="1.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="computing_query.xsd">
  <queryId> imatinib_2 </queryId>
  <clientId> 124568 </clientId>
  <date> 2018-07-11T13:45:30 </date>
  <language> en </language>

  <admin> [...] </admin>

  <drugTreatment>
    <patient>
      <covariates> [...] </covariates>
    </patient>
    <drugs> [...] </drugs>
  </drugTreatment>
  <requests> [...] </requests>
</query>
```

---

<sup>1</sup> For a complete input specification, see the file "Tucuxi CLI Software Usability Specification"

At this point, the noticeable addition is the **admin** element. This new element should contain all administrative information such as:

- Patient and adjustment mandator:
  - Title, first name, last name.
  - Address.
  - Phone .
  - Email.
- Institute of the patient and institute of the adjustment mandator:
  - Name.
  - Address.
  - Phone
  - Email.
- Miscellaneous clinical data

This administrative information will be used to find out who the patient is, who the adjustment mandator is and how they can be contacted. Finally, it should be displayed at the beginning of the report generated to know who is involved.

#### Checks:

At this stage, no check is performed.

### 4.3.2 Covariates

The covariates element contains the list of the patient's known covariates defined by an identifier **covariateId**, a **date** of measure, a **value**, a value type (**datatype**) and a **unit**.

#### Example of covariate:

```
<covariates>
  <covariate>
    <covariateId> bodyweight </covariateId>
    <date> 2018-07-11T10:45:30 </date>
    <value> 40 </value>
    <unit> kg </unit>
    <dataType> double </dataType>
  </covariate>
  [...]
</covariates>
```

#### Checks:

Covariates values are checked using drug file covariates domain.

**What is a covariate domain?**

*After a drug file is selected, each patient covariate that does not meet the domain of the same covariate in the drug file will generate a warning in the final report.*

### 4.3.3 Drugs

The drugs element contains **drug** elements. It represents the treatments the patient is undergoing. A **drug** element typically contains the associated drug identifier (**drugId**), the name of the active principle (**activePrinciple**), the manufacturer's brand name (**brandName**), the drug **atc**, the patient's **treatment**, the patient's blood **samples** and the **targets** the patient must reach.

### Example of drug:

```
<drugs>
  <drug>
    <drugId> rifampicin </drugId>
    <activePrinciple> something </activePrinciple>
    <brandName> somebrand </brandName>
    <atc> something </atc>
    <treatment> [...] </treatment>
    <samples> [...] </samples>
    <targets> [...] </targets>
  </drug>
  [...]
</drugs>
```

### Checks:

The drug identifier must match to at least one drug file.

*If no drugs match at least one drug file, the execution is aborted and an error is returned.  
If some drugs match at least one drug file, the execution continues for those drugs only.*

## 4.3.4 Dosage

The dosages are in the treatment element. It contains the patient's dosage history for a given drug. A dosage has a start date and an end date. It shows what the patient takes, when and on what basis.

The dosage element is complex but flexible. It allows describing dosages such that "take a drug at 8:00 every day except on Sunday." The main point is that it will always contain a **dose** element that allow the dosage validation.

### Example of dosage:

```
<treatment>
  <dosageHistory>
    <dosageTimeRange>
      <start> 2018-07-06T08:00:00 </start>
      <end> 2018-07-08T08:00:00 </end>
    <dosage>
      [...]
      <dose>
        <value> 400 </value>
        <unit> mg </unit>
        <infusionTimeInMinutes> 60 </infusionTimeInMinutes>
      </dose>
      [...]
    </dosage>
  </dosageTimeRange>
</dosageHistory>
  [...]
</treatment>
```

### Checks:

Each dose will be converted to match the unit of the available doses of the drug file. Then, each value will be compared to the domain of the available doses from the drug file.

*If a dose reaches the minimum or maximum bounds from the drug file, a warning will be printed in the final report.*

## 4.3.5 Samples

The samples element contains the patient's blood samples. In other words, a list of drug concentration measurements. A **sample** is defined by an identifier (**sampleId**), a date of measure (**sampleDate**) and some **concentrations**. There are multiple concentrations when the drug contains multiple analytes. Therefore, a **concentration** contains its associated analyte identifier (**analyteId**), a **value** and a **unit**.

### Example of sample:

```
<samples>
  <sample>
    <sampleId> 123456 </sampleId>
    <sampleDate> 2018-07-07T06:00:00 </sampleDate>
    <concentrations>
      <concentration>
        <analyteId> imatinib </analyteId>
        <value> 0.7 </value>
        <unit> mg/l </unit>
      </concentration>
    </concentrations>
  </sample>
  [...]
</samples>
```

### Checks:

What about a priori also?

In order to check the samples, the program will compute an "a posteriori" estimation. Then, it will check if a sample is below or above a certain percentile.

*If a sample is below the percentile X or above the percentile Y, the program will print a warning in the final report.*

## 4.3.6 Target

The adaptation engine uses targets to adapt the dosage to the patient's needs. The drug files provide such targets, but they correspond to the typical patient. In other words, those targets are generic and not patient specific. Therefore, it is possible to replace them by providing some in the query file. A target contains a corresponding active moiety (**activeMoietyId**), a type (**targetType**), a **unit** and some thresholds: minimum to reach (**min**), maximum to reach **max**, **best** to reach, inefficiency limit (**inefficacyAlarm**) and toxicity limit (**toxicityAlarm**).

### Example of target:

```
<targets>
  <target>
    <activeMoietyId> imatinib </activeMoietyId>
    <targetType> residual </targetType>
    <unit> mg/l </unit>
    <min> 20 </min>
    <best> 25 </best>
    <max> 30 </max>
    <inefficacyAlarm> 15 </inefficacyAlarm>
    <toxicityAlarm> 50 </toxicityAlarm>
  </target>
  [...]
</targets>
```

### Checks:

For a personalized target to be valid, it must have the same active moiety id as the drug file, but it must not have an identical active moiety id and target type to another personalized target.

*If a target is invalid, the program will abort and return a specific error.*

## 4.3.7 Request

The **request** element represents the computation that the core must perform. This is the way the user can control Tucuxi computing core. It contains an identifier (**requestId**) used in the response to identify the answered request, the identifier of the associated drug (**drugId**), a drug model to use (**drugModelId**) and a trait that tells the core to compute a prediction, an adjustment or some percentiles.

### Example of request:

```
<requests>

  <requestXpert> [...] </requestXpert>

  <request>
    <requestId> aposteriori </requestId>
    <drugId> imatinib </drugId>
    <drugModelId> ch.tucuxi.imatinib.gotta2012 </drugModelId>
    <COMPUTING TRAIT COMES HERE>
    </COMPUTING TRAIT COMES HERE>
  </request>
  [...]
</requests>
```

The last change from the original query format occurs here. TuberXpert introduces its own request element under the name of **requestXpert**. It is not possible to extend the original **request** with a custom trait because it contains some mandatory fields that should not be filled in by the users of TuberXpert. In the same way that a **request** controls Tucuxi computing core, a **requestXpert** will control TuberXpert.

At least, it should contain the following information:

**Should we have a more strict definition here?**

- The identifier of the drug to adjust.
- Some optional options like: Is the loading dose allowed? Is the rest period allowed?



- The type of output
  - Language (EN, FR ...).
  - Type (XML, HTML, PDF ...).
- The type of computation
  - Local. **Mmh... Not sure we should have this here**
  - Distant (currently unsupported but in anticipation of future needs).

#### Checks:

On **request** elements: no verification is performed. The system ignores them because it will make its own requests.

On **requestXpert** elements: it is supposed to respect the XML schema definition. Nevertheless, it will check that the type of output is supported before continuing the execution.

## 4.4 Program execution flow

This chapter presents the execution steps when running the program. The colors represent the main steps that are detailed in specific subchapters.

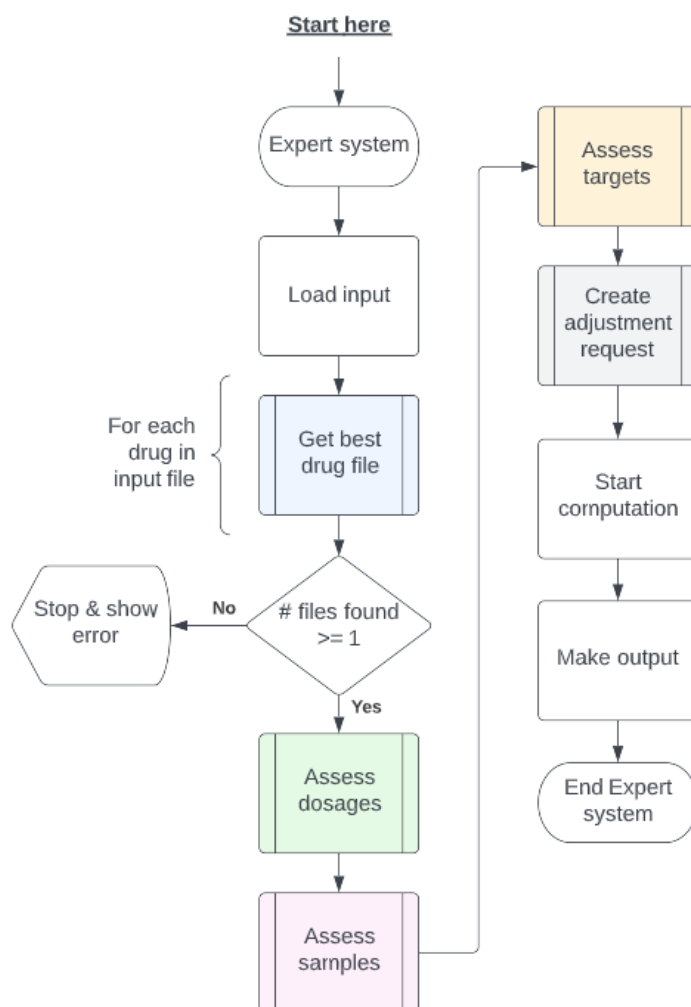


Figure 4 Program global execution

The first step is to load the query input file to manipulate it. Then, for each drug found, it tries to **get the best drug file**. If the program does not find a drug file for any drug, the execution cannot continue. Otherwise, if a drug file is found for some drugs, it assesses the **dosages**, **samples** and **targets**. After that, it creates an **adjustment request** for Tucuxi computing core. When the computation is done, it retrieves an XML response file from the computation core. Finally, it formats all the useful information into the required output. **You won't use an intermediate XML file, no?**

### 4.4.1 Get best drug file

One task of the system is to choose a drug file to use. To do this, a simple heuristic will be implemented.

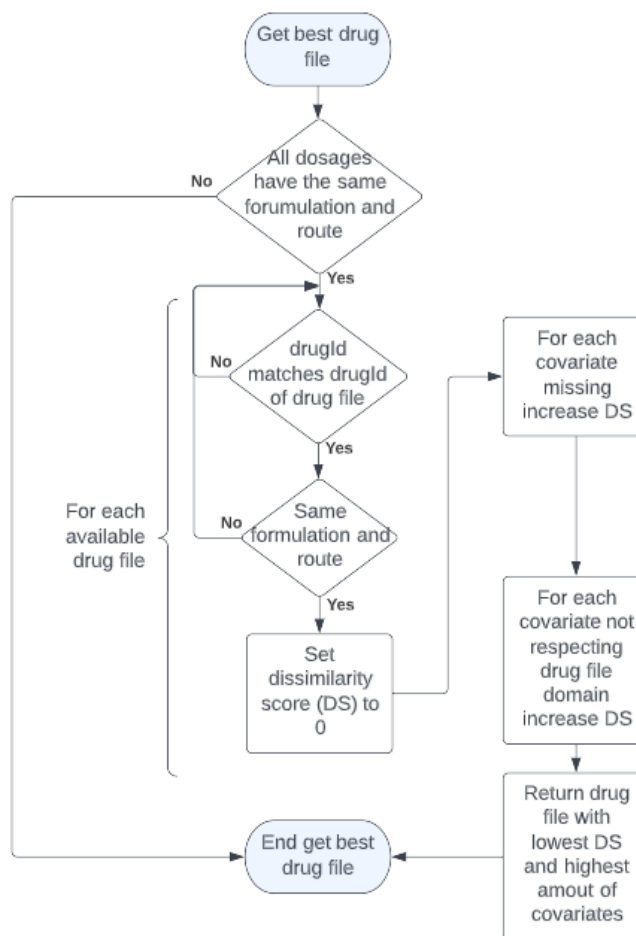


Figure 5 Process of drug files evaluation

#### Heuristic behavior

For each available model of a given drug that corresponds to the formulation and route of the dosages, we calculate a dissimilarity score  $S$  based on the covariates of the model.

$$S = \sum \text{missing covariates} + \sum \text{covariates not respecting constraints}$$

The model with the lowest dissimilarity score is chosen. In case of a tie, the model with the most covariates is chosen.

The method is not optimal, but it is a good starting point. What happens if a covariate is mandatory or if two models tie perfectly, but one may be fitting better? This type of question is not considered by this algorithm. In the future, a close collaboration with doctors would be necessary to determine for each drug, “how to choose the drug file that fit the most to the patient”.

At this point, we can say:

If the input dosages have the same formulation and route:

- Yes: Look for the best drug file.
- No: Return an error or a warning in the report.

We use the drug file name to find the drug id without loading the entire file.

In terms of architecture it would be good to have a class that could be substituted by another one in case we want to change this policy

If there are some potential drug files for each drug of the query:

- Yes: Load them.
- No: Return an error if there are none for each drug, else just an error in the final report.

In regards of the selected drug file, are the covariates supported:

- Yes: No problem.
- No: Generate a warning in the report.

#### 4.4.2 Assess the dosages

To adjust a dosage, it is important that the dosages used to perform the computation are relevant. Thus, after loading the best drug file, the CDSS will check that every dosage of the request matches the dosage domain in the drug file.

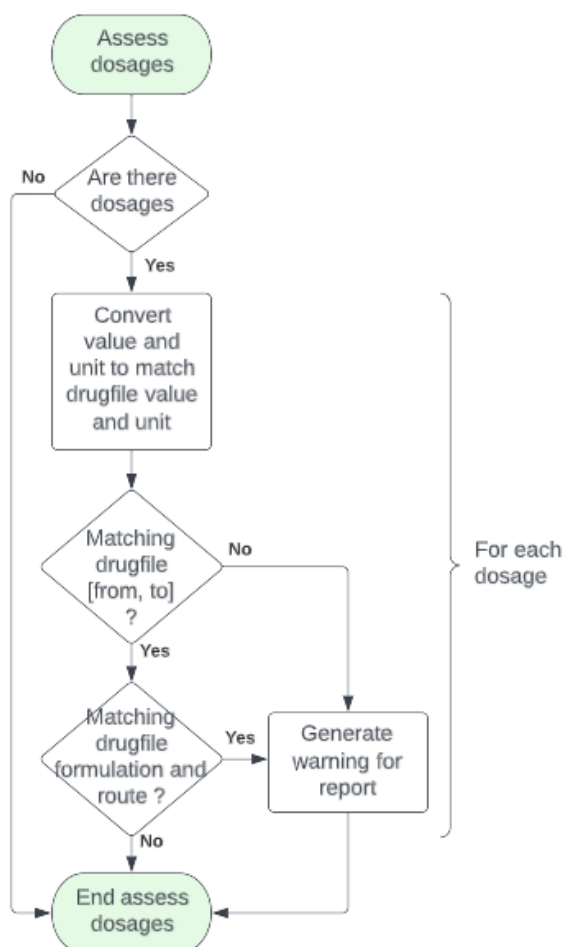


Figure 6 Process of dosages assessment

Every drug file has an availableDoses element that has a unit and a range [from, to]. This information is used to assess the dosages of the input. The Tucuxi computing core UnitManager class can convert a value from a base unit into a target unit. Thus, the CDSS will be able to convert each dosage and checks if they are in the domain of the drug file.

At this point, we can say, if the dosages are normal, i.e., within the normal range:

- Yes: No problem.
- No: Generate a warning in the report.

### 4.4.3 Assess the samples

The samples need to be representative of what was really measured. If the samples are wrong, the computation core will compute a wrong adjustment because the samples are not representative of the response of the body to the treatment.

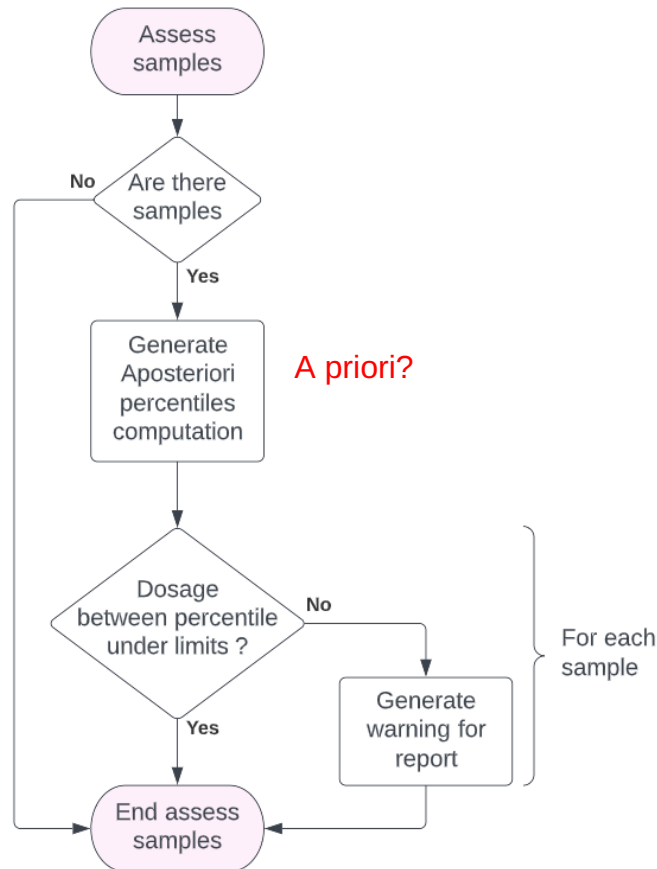


Figure 7 Process of samples assessment

Using the computation core, the system will generate an “a posteriori” percentile request. Considering the patient’s covariates, dosages and samples, it will be possible to determine which percentile the patient is in.

For example, we will use 4 percentiles that will change the level of warning:

- Below 5 or above 95: a major warning.
- Below 10 or above 90: a normal warning.

*At this point, we can say:*

*If the samples are normal, i.e., above and below a certain percentile:*

- Yes: No problem.
- No: Generate a warning in the report.

#### 4.4.4 Assess the targets

In the input XML file, it is possible to create custom targets that override the default targets of the drug file.

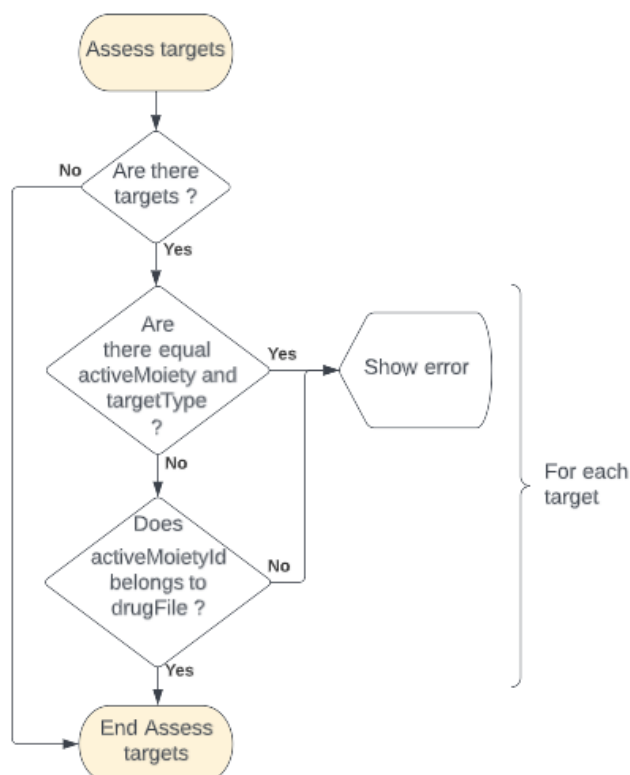


Figure 8 Process of targets assessment

Firstly, it checks if two custom targets have the same active moiety and the same target type. In this case, the targets are redundant. Since we cannot choose between these targets, we display an error and stop the adjustment for the relevant drug. Then it checks that the active moiety of the custom target is an active moiety of the drug file.

At this point, we can say:

If the custom targets are normal, i.e., non-redundant and using a good active moiety of the related drug:

- Yes: Keep them.
- No: Stop the relevant drug computation and set the return code to inform that some drugs could not be adjusted because some targets were not correct.

The possibility to write custom targets is normally intended for experienced practitioners.

There is no absolute rule that is easy to implement to determine what constitutes a relevant target for every drug. As a result, this version of the CDSS does not check for unit, min, max, best, and alarms values.

In future versions, specific rules should be implemented for each drug.

#### 4.4.5 Create adjustment request

The last step before launching an adjustment computation is to take the last decisions to prepare the request for Tucuxi computing core.

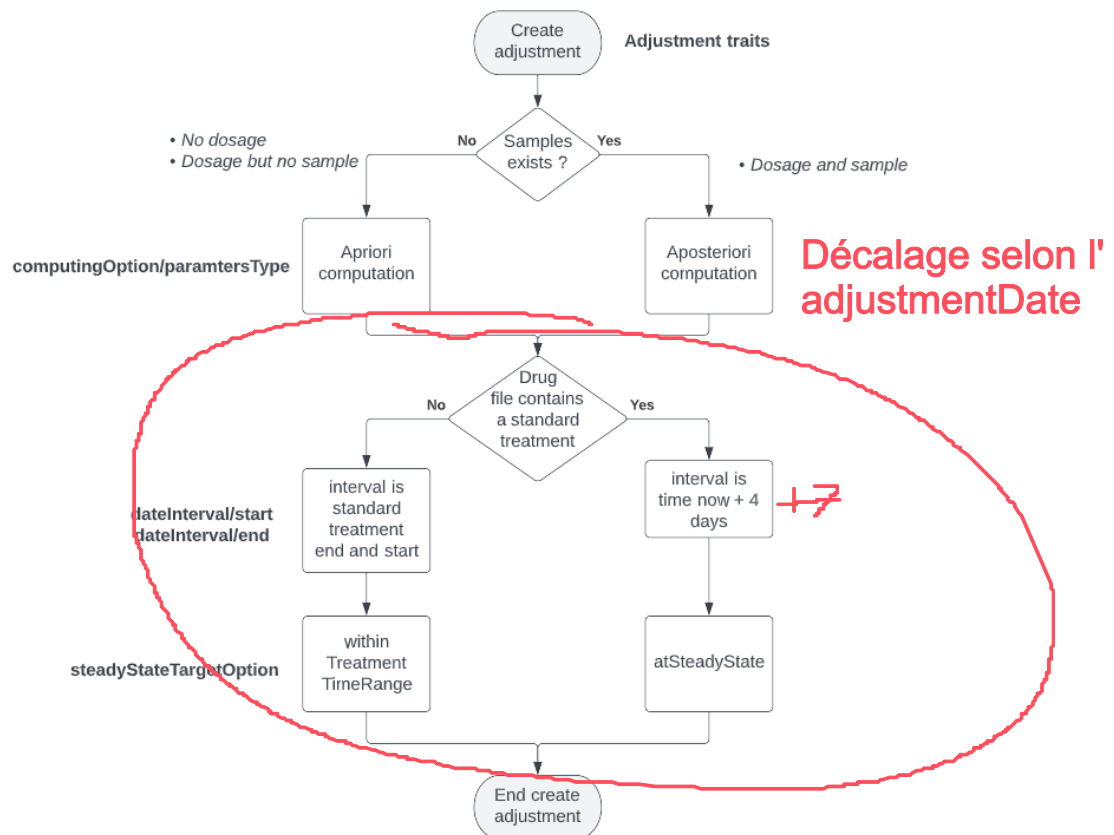


Figure 9 Decisions for adjustment request.

Three information are needed:

- What is the parameters type of the computing option?
  - If the patient has no dosage or sample, the parameters type is "a priori".
  - If the patient has a dosage and a sample, the parameters type is "a posteriori".
- What is the interval to display?
  - If the drug model does not specify a standard treatment, the interval starts today and ends in 7 days.
  - If the drug model specifies a standard treatment, the interval starts and ends accordingly to the standard treatment interval.
- What is the steady state target option?
  - If the drug model does not specify a standard treatment, the option is *atSteadyState*.
  - If the drug model specifies a standard treatment, the option is *withinTreatmentTimeRange*.

Once the program gets these answers, it can create a request with an adjustment trait. Some elements are forced by TuberXpert, but some others could be tweaked by the user using the TuberXpert custom request.

For example, here is an example of an adjustment request for Tucuxi computing core:

```
<request>
  <requestId> adjustment </requestId>
  <drugId> rifampicin </drugId>
  <drugModelId> ch.tucuxi.rifampicin.svensson2017.tdd </drugModelId>
  <adjustmentTraits>
    <computingOption>
      <parametersType> a posteriori </parametersType>
      <compartmentOption> allActiveMoieties </compartmentOption>
      <retrieveStatistics> true </retrieveStatistics>
      <retrieveParameters> true </retrieveParameters>
      <retrieveCovariates> true </retrieveCovariates>
    </computingOption>
    <nbPointsPerHour> 20 </nbPointsPerHour>
    <dateInterval>
      <start> 2018-01-12T07:00:00 </start>
      <end> 2018-03-15T12:59:00 </end>
    </dateInterval>
    <adjustmentDate> 2018-01-12T07:00:00 </adjustmentDate>
    <options>
      <bestCandidatesOption> bestDosagePerInterval </bestCandidatesOption>
      <loadingOption> noLoadingDose </loadingOption>
      <restPeriodOption> noRestPeriod </restPeriodOption>
      <steadyStateTargetOption> atSteadyState </steadyStateTargetOption>
      <targetExtractionOption> definitionIfNoIndividualTarget
                                </targetExtractionOption>
      <formulationAndRouteSelectionOption> lastFormulationAndRoute
                                </formulationAndRouteSelectionOption>
    </options>
  </adjustmentTraits>
</request>
```

The values are defined as follows:

Element	Value inside
<i>request</i>	Constant. For example, "adjustment_" + < associated drug ID>.
<i>drugId</i>	Extracted from TuberXpert request element.
<i>drugModelId</i>	Retrieved by the TuberXpert drug model selection heuristic.
<i>parametersType</i>	From decisions presented previously.
<i>compartmentOption</i>	Always <i>allActiveMoieties</i> .
<i>retrieveStatistics</i>	Always <i>true</i> . To be displayed in the final report.
<i>retrieveParameters</i>	Always <i>true</i> . To be displayed in the final report.
<i>retrieveCovariates</i>	Always <i>true</i> . To be displayed in the final report.
<i>nbPointsPerHour</i>	Always 20.
<i>start</i>	From decisions presented previously.

<i>end</i>	From decisions presented previously.
<i>adjustmentDate</i>	By default, adjust at the next approximated interval or retrieved from the TuberXpert request element.
<i>bestCandidatesOption</i>	Always <i>bestDosagePerInterval</i> .
<i>LoadingOption</i>	By default, follow drug model recommendation or retrieved from the TuberXpert request element.
<i>restPeriodOption</i>	By default, follow drug model recommendation or retrieved from the TuberXpert request element.
<i>steadyStateTargetOption</i>	From decisions presented previously.
<i>targetExtractionOption</i>	By default, <i>definitionIfNoIndividualTarget</i> or retrieved from the TuberXpert request element.
<i>formulationAndRouteSelectionOption</i>	By default, <i>lastFormulationAndRoute</i> or retrieved from the TuberXpert request element.


## 4.5 Output

This chapter discusses the forms that the output will take. It is expected to be in the form of an XML document, an HTML page or a PDF document. As a first approach to understanding what information need to be displayed, I have produced a first draft of the HTML report page on Figma. From that point, it is possible to emphasis what information is necessary and deduce what will be inserted in the XML document.

### 4.5.1 Header

This first part contains the date of generation as well as general facts about the drug concerned, such as its identifier, the last dose administered, and the drug model selected for this adjustment.

HTML representation:



# TuberXpert report

Generated on: 08.04.2022

---

**Drug:** .

**Id:** rifampicin    **Last dose:** 800 mg    **drug model:** ch.tucuxi.rifampicin.svensson2017

XML possible organization:

```

<generationDate> 2022-08-04T07:00:00 </generationDate>
<drug>
  <drugId> rifampicin </drugId>
  <lastDose>
    <value> 800 </value>
    <unit> mg </unit>
  </lastDose>
  <drugModelId> ch.tucuxi.rifampicin.svensson2017 </drugModelId>
</drug>

```

Add the computation time ?  
Computation time = generation  
date ?

Temporary renamed  
on computation time





## 4.5.2 Administrative



It contains all the administrative data of the mandator, the patient and the clinical information. In fact, this part displays every administrative data found in the admin element of the input.

HTML representation:

### Contacts


**Mandator**


**Patient**

	<b>Id:</b> - <b>Name:</b> <b>Address:</b> Av. de l'Ours2 1010 Lausanne Vaud Suisse <b>Phone:</b> 000 00 00 00 (pro) <b>Email:</b> xx@xx.xx (private)	XXX.XXXX.XXXX.XX.XX Dr. John Doe Av. de l'Ours2 1010 Lausanne Vaud Suisse 000 00 00 00 (pro) xx@xx.xx (private)
	<b>Name:</b> CHUV <b>Address:</b> Av. de l'Ours2 1010 Lausanne Vaud Suisse <b>Phone:</b> 000 00 00 00 (pro) <b>Email:</b> xx@xx.xx (private)	EHNV Av. de l'Ours2 1010 Lausanne Vaud Suisse 000 00 00 00 (pro) xx@xx.xx (private)

*Ajout de l'id*

### Clinical data

A camel case key formatted	A value.
Another key	Another value.

XML possible organization:

The XML output should stick to the structure of the admin element from the query.

More precisions will be given in the “implementation” chapter, but the structure should be organized as follows:

```

<admin>
  <mandator>
    <person> [...] </person>
    <institute> [...] </institute>
  </mandator>
  <patient>
    <person> [...] </person>
    <institute> [...] </institute>
  </patient>
  <clinicalData> [...] </clinicalData>
</admin>

```

### 4.5.3 Covariates and checks

This section lists all the covariates that are needed for the adjustment computation. It indicates the value and the unit that will be used and the source of the covariate whether it is from the patient or from the drug model.

In addition, if a covariate does not respect the drug model checks, a warning is displayed.


HTML representation:

### Covariates

<b>Fat-Free Mass</b>	<b>Value: 70 kg (default)</b>
The fat-free mass of the patient	

<b>Fat-Free Mass</b>	<b>Value: -10 kg (patient)</b>
The fat-free mass of the patient.	


The fat-free mass shall be positive.

Here, there is an example of a “fat-free mass” covariate. There is an alternative representation with a warning if the value from the patient does not meet the requirements.

XML possible organization:

```

<covariates>
  <covariate warning='Error message'>
    <name> Fat-Free Mass </name>
    <value> -10 </value>
    <unit> kg </unit>
    <desc> The fat-free mass of the patient </desc>
    <source> default / patient </source>
  </covariate>
  [...]
</covariates>

```

We could add the covariateId in case it could be useful once

Added the date when the covariate is issued form the patient

The output will return the list of the covariates used, with the following information:

- Name of the covariate
- Value entered for computation
- Unit entered for computation
- Description of the covariate
- Source of the covariate (drug model or patient query)

There is an optional attribute that may or may not be present depending on whether the covariate is within the boundaries of the drug model.

- If the covariate is out of bound, it will receive the “warning” attribute with an error value.
- Otherwise, the attribute is not added.

We should maybe think about the warnings, as we do not really use attributes...

#### 4.5.4 Treatment and checks

This section lists the dosages from the patient's dosage history. It shows each dosage within a dosage time range. It displays a warning for a dosage if the dose recommended by the drug model is reached.

HTML representation:

##### Treatment

**From:** 2018-07-06 08:00:00 **To:** 2018-07-08 08:00:00

- 100 mg daily at 10:00:00

**From:** 2018-07-08 08:00:00 **To:** 2018-07-18 08:00:00

- 10'000 mg every monday



Maximum recommended dosage reached ( 1'400 mg)

- 1200 every 12:00:00

The main idea is to display an indication of the posology near each dosage depending on their type:

- Lasting: Every + <interval>
- Daily: Daily at + <time>
- Weekly: Every <day> + at + <time>

If the dosage is inside a repeated dosage, it may be prefixed by the number of intakes. For example

- 4 x Every 12 hours

XML possible organization:

The output will return the treatment node as it entered with a small difference. Each suspicious lasting/daily/weekly dosage node will receive a "warning" attribute with an error value.

For example, the following situation could be possible:

```
<treatment>
  <dosageHistory>
    <dosageTimeRange>
      <start> 2018-07-06T08:00:00 </start>
      <end> 2018-08-08T08:00:00 </end>
      <dosage>
        <dosageLoop>
          <lastingDosage warning='Error message'>
            [...]
          </lastingDosage>
        </dosageLoop>
      </dosage>
    </dosageTimeRange>
  </dosageHistory>
</treatment>
```

### 4.5.5 Samples and check

This section lists the patient's samples. It shows the date of the sample, its measure and the percentile to which it belongs. It displays a warning for a sample if it reaches some given threshold:

- Red warning if the percentile is below 5 or above 95
- Yellow warning if the percentile is below 10 or above 90

HTML representation:

#### Samples

**Date:** 2018-07-06 08:00:00

**Mesure:** 7mg/l

**Percentile:** 50

**Date:** 2018-07-07 08:00:00

**Mesure:** 700 mg/l

**Percentile:** 100



100% of the population is below this measure

**Date:** 2018-07-08 08:00:00

**Mesure:** 2 mg/l

**Percentile:** 8



92% of the population is above this measure

XML possible organization:

The output will return the samples node as it entered with two differences. Each suspicious concentration will receive a "warning" attribute with an error value and each concentration will receive a new "percentile" element.

For example, the following situation could be possible:

Pas implémenté dans  
sample

```
<samples>
  <sample>
    <sampleId> sample_1 </sampleId>
    <sampleDate> 2018-07-06T08:00:00 </sampleDate>
    <concentrations>
      <concentration warning='Error message'>
        <analyteId> rifampicin </analyteId>
        <percentile> 50 </percentile>
        <value> 7 </value>
        <unit> mg/l </unit>
      </concentration>
      [...]
    </concentrations>
  </sample>
  [...]
</ samples >
```

The "warning" attribute is optional. So, it won't be used for expected concentration.

## 4.5.6 Adjustments

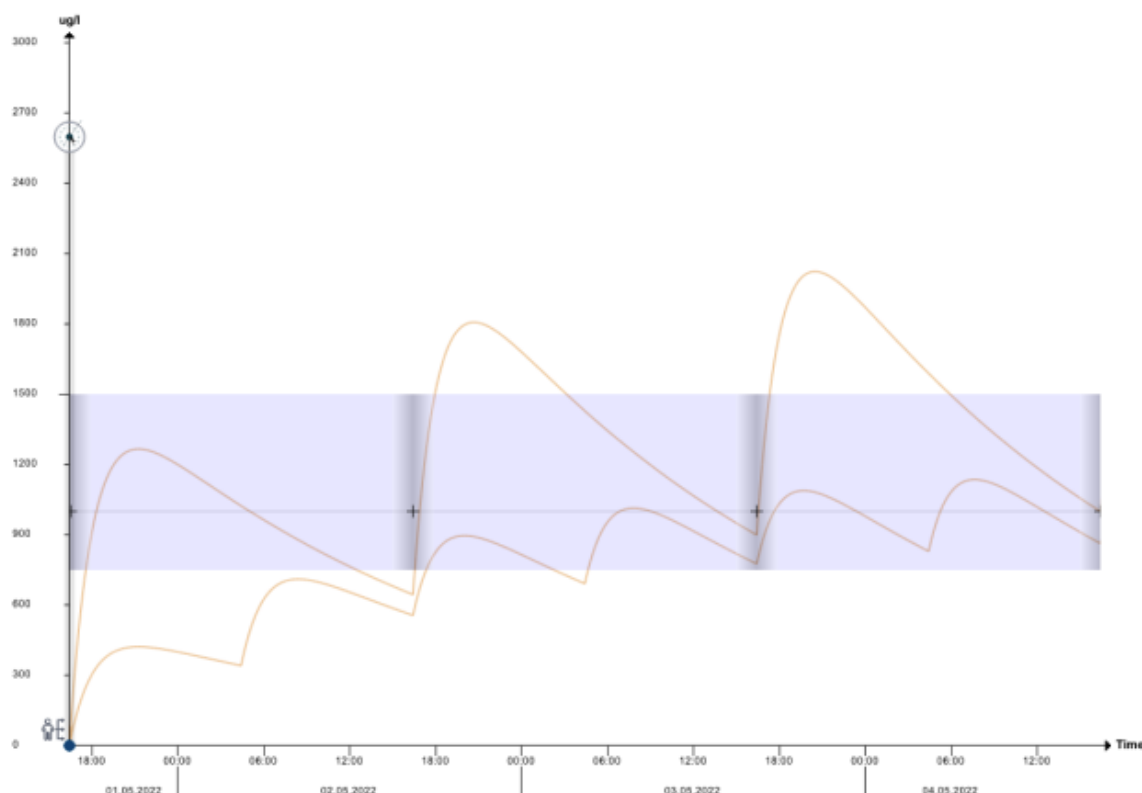
This section shows the possible adjustments and the one that is judged the best by Tucuxi computation core.

HTML representation:

### Adjustment

Based on the above information, the following adjustments are relevant to achieve the targets.

#### Per interval



#### Displayed adjustments

	From	To	Dose	Interval	Duration	Score
●	01.05.2022 16:25	06.05.2022 16:25	600 mg	24 hours	63	0.95
	Loading dose: 800 mg on 01.05.2022 16:25					
●	01.05.2022 16:25	06.05.2022 16:25	200 mg	12 hours	125	0.87

This first part of the adjustment section displays a short introductory sentence followed by a graph of all the adjustments found. Below the graph, a table lists all the displayed dosages. It indicates the time range, the dose, the intake interval, the number of intakes and the attributed score of adjustment quality by Tucuxi computation core. For a complete specification of the score, see the document “Tucuxi System Description”.

$$Score = 1 - \frac{(\log(C_{predicted}) - \log(C_{target}))^2}{(0.5\log(C_{up}) - 0.5\log(C_{lo}))^2}$$

Figure 10 Formula of the target score based on concentration C.

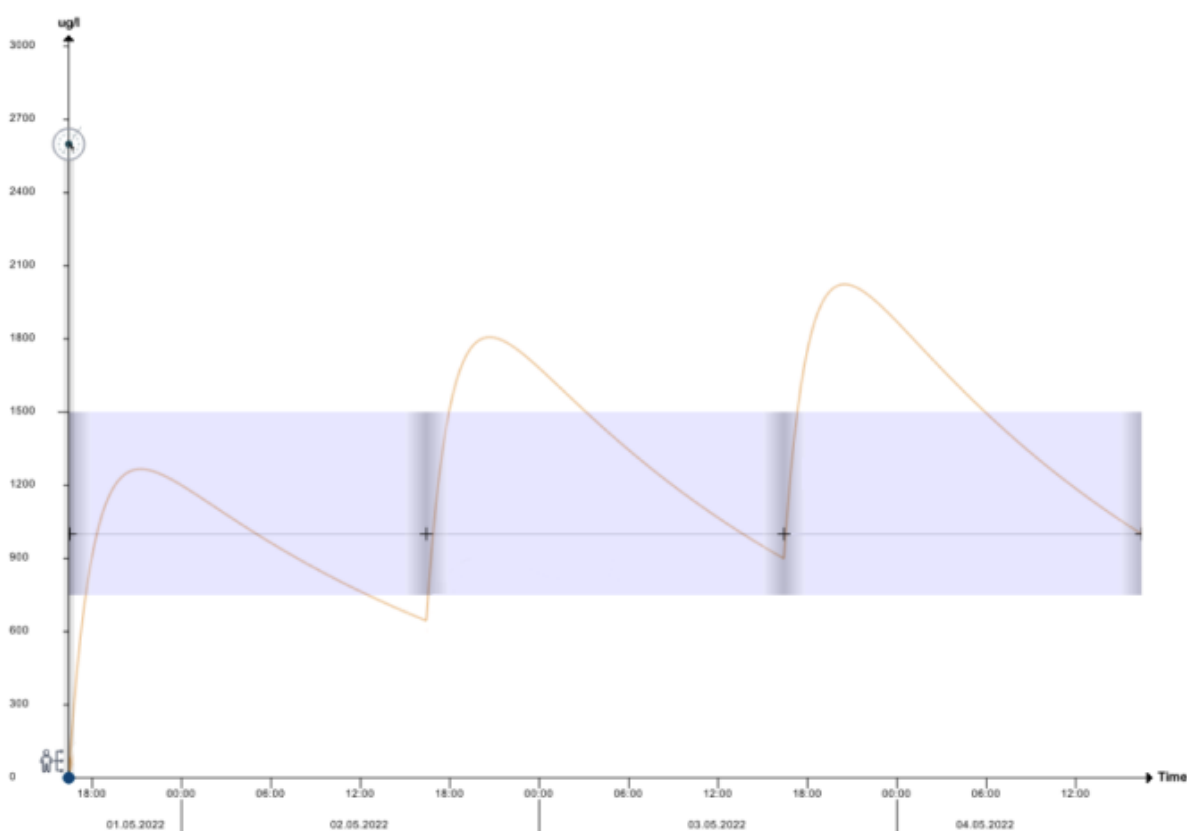
The higher the score, the better. Each target gets a score, and the adjustment score is the average of all the target scores.

In addition, if necessary, it indicates the required loading dose.

## Suggestion

Considering the above information, TuberXpert recommends the following adjustment:

	From	To	Dose	Interval	Duration	Score
●	01.05.2022 16:25	06.05.2022 16:25	600 mg	24 hours	63	0.95
Loading dose: 800 mg on 01.05.2022 16:25						



With this new treatment, the targets are expected to change as follows

	Type	value (unit)	Score
●	Residual	1103.12 (ug/l)	0.95
Inefficacy: 0 / Min:750 / Best: 1000 / Max: 1500 / Toxicity: 10000			

This second part highlights the adjustment with the highest score. It shows its time range, the dose, the intake interval, the number of intakes and the score assigned by Tucuxi computation. Additionally, it displays the achievement of the targets. For each target, it lists its type, its predicted value, its score and the thresholds.

## XML possible organization: Why possible? (The same applies for other fields)

The XML content should be roughly be the same as the “*DataAdjustment*” element from the Tucuxi computation response. For example, the following XML structure could be possible:

```
<dataAdjustment>
  <analyteIds>
    <analyteId> rifampicin </analyteId>
  </analyteIds>
  <adjustments>
    <adjustment>
      <score> 0.985 </score>
      <targetEvaluations> [See below] </targetEvaluations>
      <dosageHistory> [See below] </dosageHistory>
      <cycleDatas> [See below] </cycleDatas>
    </adjustment>
    [...]
  </adjustments>
</dataAdjustment>
```

The **analyteIds** is the list of all the analyte identifiers involved in the **cycleDatas**. It is only used when a drug contains more than one analyte.

```
<targetEvaluations>
  <targetEvaluation>
    <targetType> residual </targetType>
    <unit> ug/l </unit>
    <value> 1103.122367 </value>
    <score> 0.919806 </score>
    <min> 750 </min>
    <best> 1000 </best>
    <max> 1500 </max>
    <inefficacyAlarm> 0 </inefficacyAlarm>
    <toxicityAlarm> 10000 </toxicityAlarm>
  </targetEvaluation>
  [...]
</targetEvaluations>
```

The **targetEvaluation** element is much like a target element, but with a **value** and an achievement **score**.

```
<dosageHistory>
  <dosageTimeRange>
    <start> 2018-01-12T07:00:00 </start>
    <end> 2018-03-15T12:59:00 </end>
    <dosage> [...] </dosage>
  </dosageTimeRange>
  [...]
</dosageHistory>
```

The dosage history element contains the list of dosage to be followed for the adjustment. Its structure is the same as the one in the query dosage history.

When there are multiple **dosageTimeRange**, if the first one contains a repetition of one iteration, it is a loading dose.

```
<cycleDatas>
  <cycleData>
    <start> 2018-01-12T07:00:00 </start>
    <end> 2018-01-12T19:00:00 </end>
    <unit> ug/l </unit>
    <times> [...] </times>
    <values> [...] </values>
  </cycleData>
  [...]
</cycleDatas>
```

**CycleDatas is a list of CycleData**

A CycleDatas contains the predictions between two intakes. A cycleData contains a **start** and **end** date, the **unit** of the **values**, a **times** element that contains a comma-separated list of times, in hours, starting from zero and a **values** element that contains a comma-separated list of concentration values. The number of times matches the number of values.

### 4.5.7 Computation facts

This is the last section of the report. It contains general facts about the computation, such as the pharmacokinetics parameters, some steady-state predictions and the covariates used by the computation core. This second list of covariates is useful because it allows to double check the covariates. In addition, some covariates can be calculated on running time based on the patient's other covariates. Thus, it is possible to see if there are any calculated covariates.

#### HTML representation:

##### Pharmacokinetic parameters

	Typical patient	A priori	A posteriori
● CL	15.106	15.202	15.202
● F	1	1	1

##### Predictions

<b>Extrapolated steady state AUC24</b>	36863.6 ug*h/l
<b>Steady state peak</b>	2023.58 ug/l
<b>Steady state trough</b>	999.701 ug/l

##### Covariates used for computation

Name	value
● Bodyweight	50
● Age	40



## XML possible organization:

```
<parameters>
  <typical>
    <parameter>
      <id> Ka </id>
      <value> 0.609 </value>
    </parameter>
    [...]
  </typical>
  <apriori>
    <parameter> [...] </parameter>
    [...]
  </apriori>
  <aposteriori>
    <parameter> [...] </parameter>
    [...]
  </aposteriori>
</parameters>
<statistics>
  <auc24> 36863.6 </auc24>
  <peak> 2023.58 </peak>
  <residual> 999.701 </residual>
</statistics>
<computationCovariates>
  <computationCovariate>
    <id> bodyweight </id>
    <value> 40.000000 </value>
  </computationCovariate >
  [...]
</computationCovariates>
```

The **parameters** element contains a listing of each pharmacokinetics **parameter** for each computation type: **typical**, **apriori** and **aposteriori**. Depending on the parameters type of the request, additional adjustment requests are required to obtain all parameters. For example, if the current adjustment is “a posteriori”, additional adjustment requests are made to get the “typical” and “a priori” parameters.

The **statistics** element contains the predictions at steady state. The steady state can be approximated using the formula:

$$\text{adjustment date} + \text{half-life} * 2$$

Not really, but we have half-life and multiple

These statistics are in each cycleData returned by the Tucuxi computing core. Consequently, it is enough to carry out an adjustment request which will be able to return the corresponding cycleData.

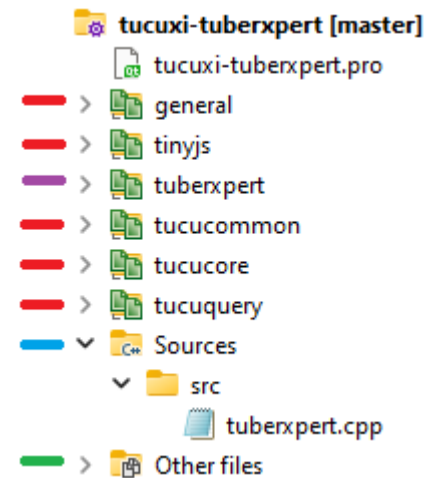
The **computationCovariates** element lists all the covariates used during the computation, represented by a **computationCovariate** element which contains a **value** and the **id** of covariate.

## 5 Implementation

This chapter presents the main steps in the implementation. Each subchapter corresponds to a component or feature. They can provide a UML class, explain their goal, provide a description of how they work or justify certain decisions.

The organization of the project is as follows:

- It includes the implementation of Tucuxi computing core, tucuxi-core.
- The implementation of TuberXpert is accessible by the “tuberxpert” inclusion.
- The master program is in a single file “tuberxpert.cpp.” It contains the main function that drives the execution.
- The “Other files” folder contains some XML files and some validation files used by the query importer and the language manager.



### 5.1 Query importer

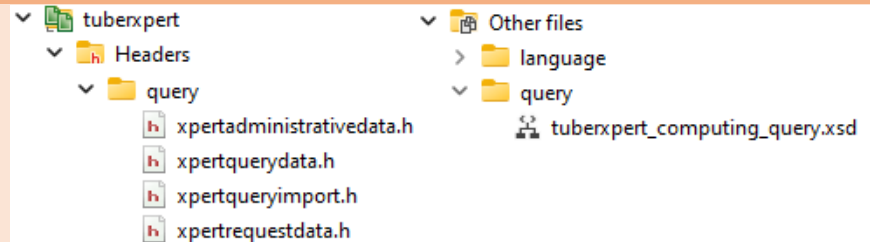
#### Goal

#### of importing

#### Location

This component is in charge to import the XML query file in TuberXpert.

The structure of the XML query file can be validated using the file “tuberxpert\_computing\_query.xsd.”



#### 5.1.1 XML query structure

The starting point is to understand the structure of the XML query file. As exposed in the “Analyze / Input” chapter, the structure of the TuberXpert query file is the same as the Tucuxi computing core. The exception is that it contains two new elements:

- An admin element that contains the administrative data.
- A requestXpert element that tells TuberXpert what to do.

From the query element, the new structure is:

```
<query>
  [...]
  <admin> [...] </admin>
  [...]
  <requests>
    <requestXpert> [...] </requestXpert>
    [...]
  </requests>
</query>
```

Tag name	Format	Occ.	Description
<query>		1:1	Data about the query
_<admin>	Admin	0:1	The administrative data
_<requests>	Requests	1:1	Requests specification
__<requestXpert>	RequestXpert	1: ∞	The request for TuberXpert

*The original query elements that are not displayed are still part of it and have not been changed.*

### Admin element

Nadir Benallal

The structure of the **admin** element is inspired by the one from the bachelor's thesis of Benallal Nadir. It is flexible and contains every needed field to store contact information of the **patient** and the **mandator** as well as clinical data of any kind.

```
<admin>
  <mandator>
    <person> [...] </person>
    <institute>. [...] </institute>
  </mandator>
  <patient>
    <person> [...] </person>
    <institute>. [...] </institute>
  </patient> [...]
  <clinicalData> [...] </clinicalData>
</admin>
```

Tag name	Format	Occ.	Description
<admin>		0:1	The administrative data
_<mandator>		0:1	The mandator of the adjustment
__<person>	Person	1:1	Personal contact information of the mandator
__<institute>	Institute	0:1	Institute contact information of the mandator
_<patient>		0:1	The patient that will follow the adjustment.
__<person>	Person	1:1	Personal contact information of the patient
__<institute>	Institute	0:1	Institute contact information of the patient
_<clinicalData>	Key-Value	0:1	The clinical data

*The clinical data can contain any element with any value, but not nested elements.*

*For example, it could contain: <roomNumber> 25 </roomNumber>.*

*The name of the element should be written in camelCase in order to be properly rendered in the HTML/PDF report.*

Is that compatible with a XSD schema and validation?

### Person element

A person element contains the personal contact information of a person.

```
<person>
  <id> asdf </id>
  <title> Dr. </title>
  <firstName> John </firstName>
  <lastName> Doe </lastName>
  <address> [...] </address>
  <phone> [...] </phone>
  <email> [...] </email>
</person>
```

Tag name	Format	Occ.	Description
<person>		1:1	Personal contact information
_<id>	string	0:1	Identifier of the person
_<title>	string	0:1	Title of the person
_<firstName>	string	1:1	The first name of the person
_<lastName>	string	1:1	The last name of the person
_<address>	Address	0:1	Address of the person
_<phone>	Phone	0:1	Phone number of the person
_<email>	Email	0:1	Email of the person

### Institute element

An institute element contains the contact information of an institute.

```
<institute>
  <id> 456789 </id>
  <name> CHUV </title>
  <address> [...] </address>
  <phone> [...] </phone>
  <email> [...] </email>
</institute>
```

Tag name	Format	Occ.	Description
<institute>		0:1	Institute contact information
_<id>	string	0:1	Identifier of the institute
_<name>	string	1:1	Name of the institute
_<address>	Address	0:1	Address of the institute
_<phone>	Phone	0:1	Phone number of the institute
_<email>	Email	0:1	Email of the institute

### Address element

An address element contains the address information of a person or an institute.

```
<address>
  <street> Av. de l'Ours 1 </street>
  <postCode> 1010 </postCode>
  <city> Lausanne </city>
  <state> Vaud </state>
  <country> Suisse </country>
</address>
```

Tag name	Format	Occ.	Description
<address>		0:1	Address of an institute or a person
_<street>	string	1:1	Street of the address
_<postCode>	string	1:1	Postal code of the address
_<city>	string	1:1	City of the address
_<state>	string	0:1	State of the address
_<country>	string	0:1	Country of the address

### Phone element

A phone element contains a number and a type of phone number for a person or an institute.

```
<phone>
  <number> 0213140001 </number>
  <type> private </type>
</phone>
```

Tag name	Format	Occ.	Description
<phone>		0:1	Phone number of a person or an institute
_<number>	string	1:1	Phone number
_<type>	string	0:1	Phone type

*The <type> tag is a string enumeration. It can be "private" or "professional".*

### Email element

An email element contains an email address and his type for a person or an institute.

```
<email>
  <address> anemail@email.mail </address>
  <type> professional </type>
</email>
```

Tag name	Format	Occ.	Description
<email>		0:1	Email of a person or an institute
_<address>	string	1:1	Email address
_<type>	string	0:1	Email type

The <type> tag is a string enumeration. It can be “private” or “professional”.

### requestXpert element

The requestXpert element guides TuberXpert. It tells which drug to adjust, the computation type, which is the desired output, the adjustment date and some computing options.

```
<requestXpert>
  <drugId> rifampicin </drugId>
  <localComputation> true </localComputation>
  <output>
    <format> XML </format>
    <language> en </language>
  </output>
  <adjustmentDate> 2018-07-06T08:00:00 </adjustmentDate>
  <options>
    <loadingOption> noLoadingDose </loadingOption>
    <restPeriodOption> noRestPeriod </restPeriodOption>
    <targetExtractionOption> populationValues </targetExtractionOption>
    <formulationAndRouteSelectionOption> lastFormulationAndRoute
                                     </formulationAndRouteSelectionOption>
  </options>
</requestXpert>
```

Tag name	Format	Occ.	Description
<requestXpert>		1: ∞	The request for TuberXpert
_<drugId>	string	1:1	The identifier of the drug to adjust
_<localComputation>	boolean	1:1	The type of computation
_<output>		1:1	Specifications about the output
__<format>	string	1:1	Output format
__<language>	string	1:1	Output language
_<adjustmentDate>	date	0:1	Date of adjustment
_<option>		0:1	Options specifications
__<loadingOption>	string	0:1	Allow loading dose or not
__<restPeriodOption>	string	0:1	Allow rest period or not
__<targetExtractionOption>	string	0:1	Extraction option for targets

<code>&lt;formulationAndRouteSelectionOption&gt;</code>	string	0:1	Selection of the potential formulations and routes
---	--------	-----	--

The tag `<localComputation>` allows choosing whether the computation is:

- Local: true
- Remote: false (not yet implemented)

Sometimes the computers do not have enough computing power and send the computation to a distant server. This option is not part of the current implementation. **Still, not sure it makes sense**

The `<format>` tag is a string enumeration that allows choosing the output format. It can be "html", "xml" or "pdf".

The `<language>` tag is a string enumeration that allows choosing the output language. It can be "en" or "fr".

The `<loadingOption>` tag is a string enumeration. It can be "noLoadingDose" or "loadingDoseAllowed".

From Tucuxi CLI Software Usability Specification:

- "noLoadingDose: No loading dose can be added to the new dosage"
- "loadingDoseAllowed: If the current dosage is under the target, a loading dose can be added at the beginning of the new dosage to more rapidly reach the optimum."

If the tag is not present, the recommendation from the drug model is used.

The `<restPeriodOption>` tag is a string enumeration. It can be "noRestPeriod" or "restPeriodAllowed".

From Tucuxi CLI Software Usability Specification:

- "noRestPeriod: No rest period can be added to the new dosage"
- "restPeriodAllowed: If the current dosage is over the target, a rest period can be added at the beginning of the new dosage to more rapidly reach the optimum."

If the tag is not present, the recommendation from the drug model is used.

The `<targetExtractionOption>` tag is a string enumeration. It can be "populationValues", "aprioriValues", "individualTargets", "individualTargetsIfDefinitionExists", "definitionIfNoIndividualTarget" or "individualTargetsIfDefinitionExistsAndDefinitionIfNoIndividualTarget".

From Tucuxi CLI Software Usability Specification:

- "populationValues: Forces the population values to be used"
- "aprioriValues: Forces the a priori values to be calculated and used"
- "individualTargets: Only use the individual targets"
- "individualTargetsIfDefinitionExists: Only use the individual targets if a target definition exists"
- "definitionIfNoIndividualTarget: Use the individual target, and if for an active moiety and a target type no individual target exists, then use the definition"
- "individualTargetsIfDefinitionExistsAndDefinitionIfNoIndividualTarget: Use the individual target if a target definition exists, and if for an active moiety and a target type no individual target exists, then use the definition"

If the tag is not present, the value "definitionIfNoIndividualTarget" is used.

The `<formulationAndRouteSelectionOption>` tag is a string enumeration. It can be “lastFormulationAndRoute”, “defaultFormulationAndRoute” or “allFormulationAndRoutes”.

From Tucuxi CLI Software Usability Specification:

- “lastFormulationAndRoute: Use only the last formulation and route used in the current treatment. If the treatment is empty, then use the default formulation and route of the drug model.”
- “defaultFormulationAndRoute: Use only the default formulation and route of the drug model”
- “allFormulationAndRoutes: Use all available formulation and routes of the drug model”

If the tag is not present, the value “lastFormulationAndRoute” is used.

## 5.1.2 Administrative data

The modeling of administrative data is in the “query/xpertadministrativedata” class.

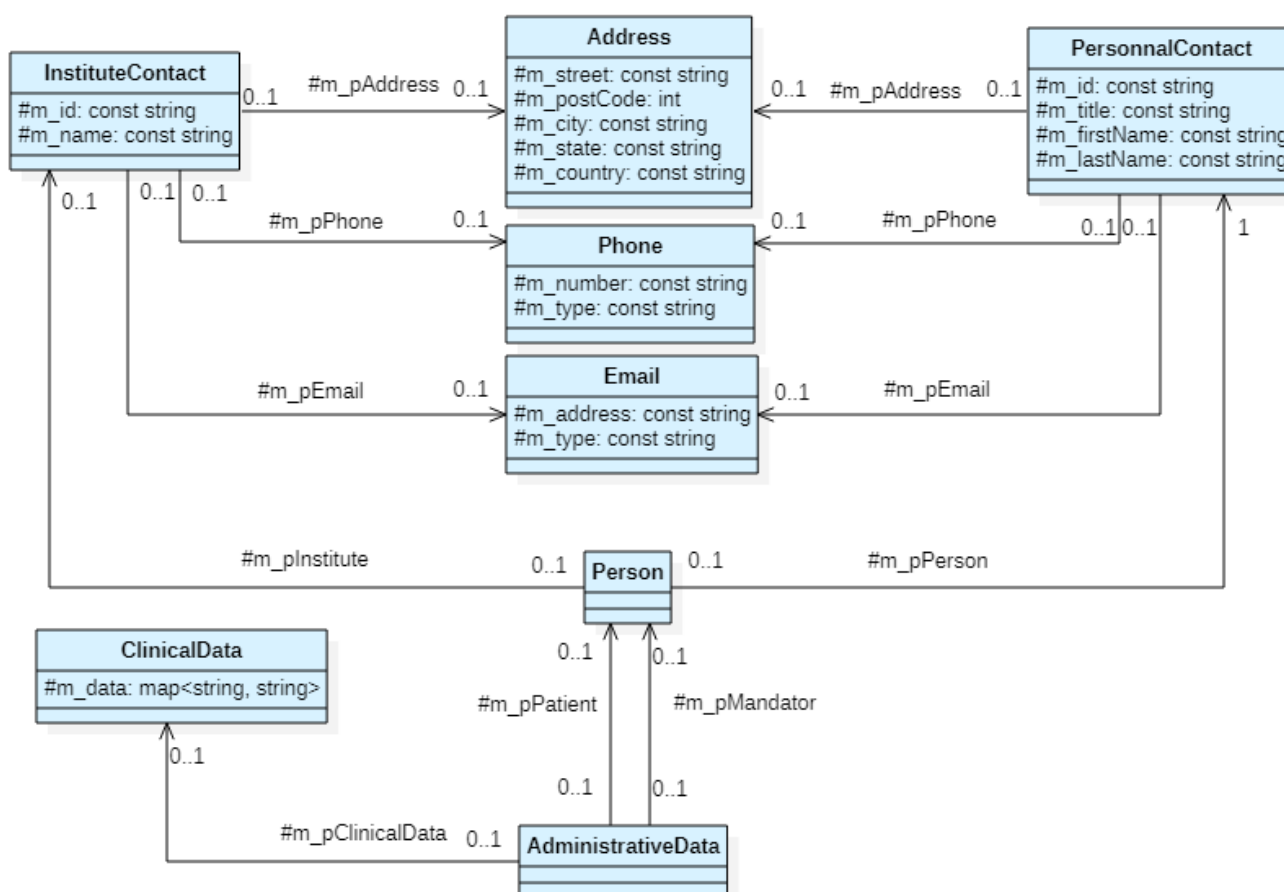


Figure 11 Class diagram of administrative data

The methods of each blue class are not displayed since they are simple getters on their attributes. They have a unique constructor that takes all their attributes.

The class modeling follows the same structure as the query file. The **admin** element is represented by the AdministrativeData class. The ClinicalData class contains a map for each key-value pair in the **clinicalData** element. The Person class is used for the **mendator** and the **patient**. The PersonalContact and InstituteContact classes contain data from the **person** and **institute** element. The Address, the Phone and Email classes encapsulate the information contained in the **address**, **phone** and **email** elements.



### 5.1.3 RequestXpert data

The modeling of TuberXpert request data is in the “query/xpertrequestdata” class.

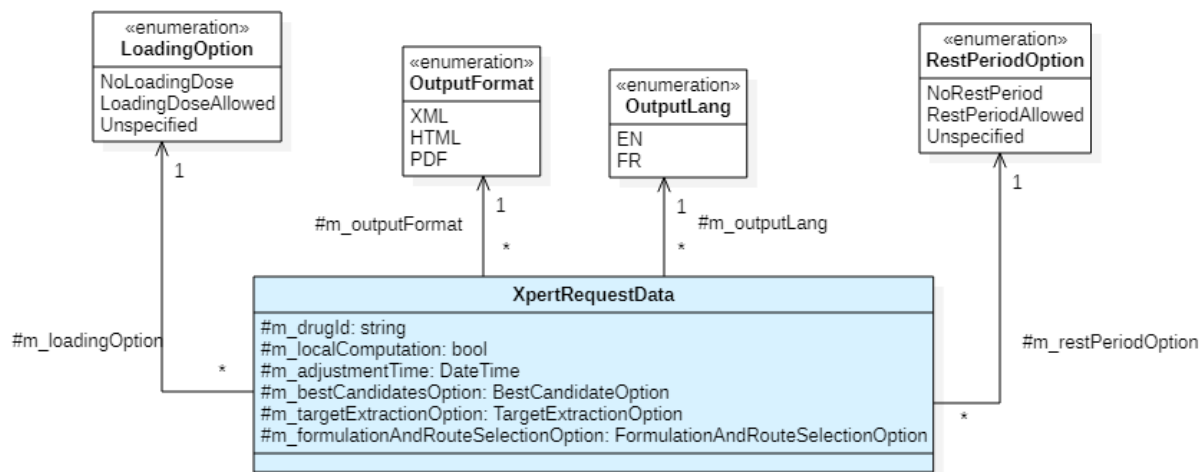


Figure 12 Class diagram of the requestXpert data

The methods of each blue class are not displayed since they are simple getters on their attributes. They have a unique constructor that takes all their attributes.

The **requestXpert** element is represented by the XpertRequestData class. The **output format** and **language** are represented by the enumerations OutputFormat and the OutputLang. The loadingOption and restPeriodOption values are translated into the LoadingOption and the RestPeriodOption enumerations. The value “Unspecified” is used when the user does not explicitly allow or disallow these options. This means that the recommendations of the drug model should be used. Finally, the targetExtractionOption and the formulationAndRouteSelectionOption use the enumerations implemented in Tucuxi computing core. Thus, the values are not displayed on this UML.

### 5.1.4 Query data

Therefore ... Seems strange at the beginning of a section

Therefore, the Tucuxi computing core query class QueryData must be specialized to incorporate the two new elements brought by TuberXpert in the query file. This is made with the “query/xpertquerydata” class.

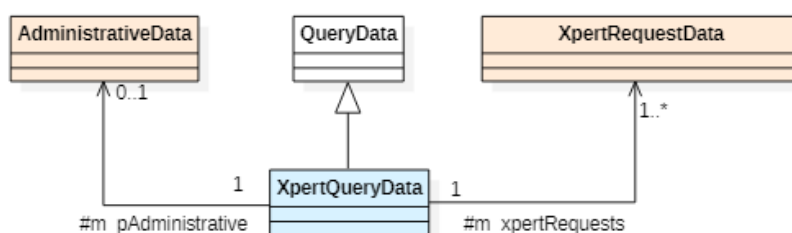


Figure 13 Class diagram of the TuberXpert query

The orange classes have already been explained. Their representation is simplified to their name.

The methods of each blue class are not displayed since they are simple getters on their attributes. They have a unique constructor that takes all their attributes.

The TuberXpert query class XpertQueryData extends the QueryData class that is used by Tucuxi computing core. It adds the administrative information and the TuberXpert requests.

### 5.1.5 Query importation

The last step to complete the importation is to implement the TuberXpert query importer “query/xpertqueryimport” class.

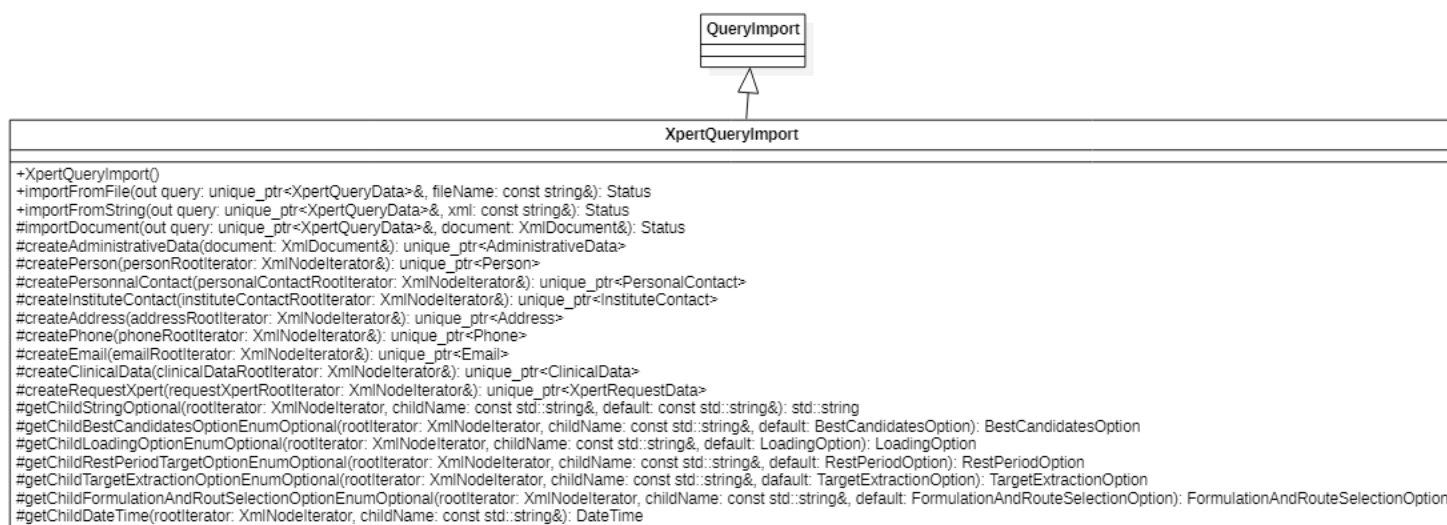


Figure 14 Class diagram of the TuberXpert query importer

The XpertQueryImport class extends the query importer of Tucuxi computation core. The public methods import a query file in an XmlDocument either by file name or by XML string and make an XpertQueryData object. The protected methods parse the document to create some corresponding objects. For example, the method createPhone will parse the phone element of the query to create a Phone object. Common elements in a query file between Tucuxi computing core and TuberXpert are imported by the methods inherited from the QueryImport class. If the importation went well, the returned value of Status is ok.

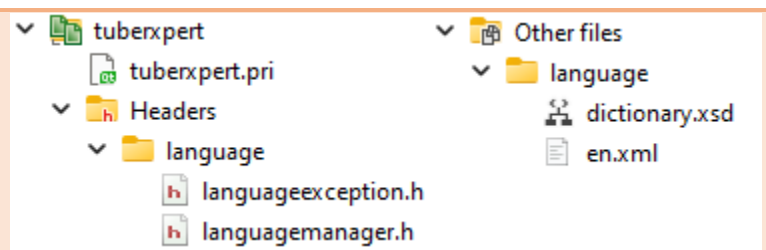
## 5.2 Language manager

### Goal

This component is in charge to import an XML language file and use it for the report generation.

The structure of the XML translation file can be validated using the file “dictionary.xsd”.

### Location



### 5.2.1 XML dictionary structure

A translation file is a list of entries. Each entry has an attribute that stores a key, and a value that is a translation.

```

<dictionary>
  <entry key="A_key" > A translation </entry>
  [...]
</dictionary>
    
```

Tag name	Format	Occ.	Description
<dictionary>		1:1	Data about the dictionary
_<entry>	String	0: ∞	A translation to retrieve thanks to the key attribute.

### 5.2.2 Getting a translation

The translation file is loaded and accessed using the LanguageManager class “language/langagemanager”.

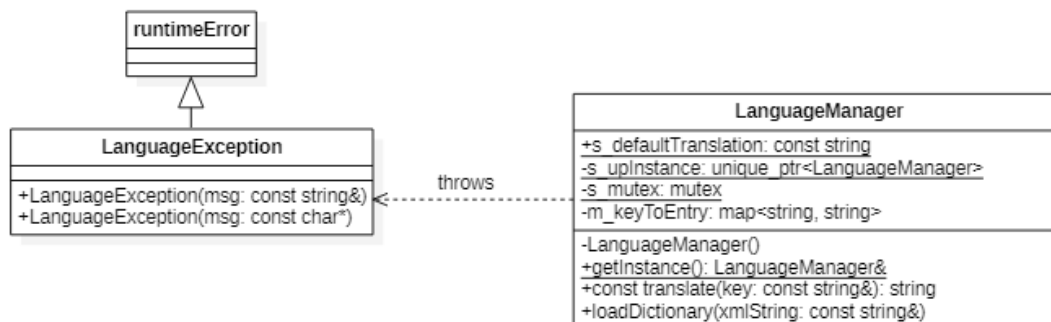


Figure 15 Class diagram of the TuberXpert language manager

The `LanguageManager` class is a singleton. When the instance is retrieved, it is possible to load an XML language file by giving its content in the method `loadDictionary`. Each contained key-value pair is stored in the map `keyToEntry`. To access a translation, the `translate` method takes a key as argument and it returns the associated value. If the key does not exist, the `defaultTranslation` string is returned. If the loading of the dictionary string fails, a `LanguageException` is thrown.

## 6 Tests

This chapter presents the tests that were performed to validate the behavior of the system.

### 6.1 Unit Tests

Why fructose?

A test program has been set up to perform unit testing. To do this, it uses the framework Fructose<sup>2</sup>.

#### 6.1.1 LanguageManager

The “language/languagemanager” class is tested by the test\_languagemanager class.

**Name:**

Retrieve dictionary

**Result:**

Success

**Description:**

The test tries to perform loadDictionary with various XML strings.

**Validation:**

- When a bad string is used, a `LanguageException` is raised.
- When a valid string is used, no exception is thrown.

**Name:**

Word translation

**Result:**

Success

**Description:**

The test tries to perform translate method calls with an invalid key and then with a valid key.

**Validation:**

- When the wrong key is used, it gets the string `Tucuxi::Language::LanguageManager::defaultTranslation`.
- When the valid key is used, the corresponding value from the dictionary XML file is retrieved.

#### 6.1.2 XpertQueryImport

The “query/xpertqueryimport” class is tested by the test\_xpertqueryimport class.

**Name:**

Retrieve complete admin

**Result:**

Success

**Description:**

The test tries to load a query file that contains the most complete admin element possible. It includes a mandator with his institute, a patient with his institute and a non-empty clinical data element. Then, it checks if all the information from the file is correctly returned.

**Validation:**

- The import status is `Status::Ok`.
- All information from the file is correctly returned.

<sup>2</sup> <http://www.andrewpetermarlow.co.uk/software/fructose.html>

**Name:**  
Retrieve no admin

**Result:**  
Success

**Description:**  
The test tries to load a query file without any admin element inside.

**Validation:**

- The import status is `Status::Ok`.
- When getting the `std::optional<std::reference_wrapper<const AdministrativeData>>`, it has no value.

**Name:**  
Retrieve empty admin

**Result:**  
Success

**Description:**  
The test tries to load a query file with an admin element that is empty.

**Validation:**

- The import status is `Status::Ok`.
- When getting the `std::optional<std::reference_wrapper<const AdministrativeData>>`, it has a value.
- When using getters of the received administrative data, the optional values returned have no value.

**Name:**  
Retrieve minimal person

**Result:**  
Success

**Description:**  
The test tries to load a query file with minimal patient and mandator. It means that they only have a first name and a last name.

**Validation:**

- The import status is `Status::Ok`.
- When getting their identifier and title, the getters return an empty string.
- When getting their address, phone, email and institute, the getters return an optional without value.

**Name:**  
Retrieve minimal institute

**Result:**  
Success

**Description:**  
The test tries to load a query file with minimal patient and mandator institute.

**Validation:**

- The import status is `Status::Ok`.
- When getting their identifier, the getters return an empty string.
- When getting their address, phone and email, the getters return an optional without value.

**Name:**

Retrieve minimal coordinates

**Result:**

Success

**Description:**

The test tries to load a query file with minimal address, phone and email in persons and institutes.

**Validation:**

- The import status is **Status::Ok**.
- When getting their missing values, it returns an empty string.

**Name:**

Error when missing mandatory in mandator person

**Result:**

Success

**Description:**

The test tries to load a query file with missing mandatory values in mandator person.

**Validation:**

- The import status is **Status::Error**.
- All the mandatory nodes appear in error message.

**Name:**

Error when missing mandatory in mandator institute

**Result:**

Success

**Description:**

The test tries to load a query file with missing mandatory values in mandator institute.

**Validation:**

- The import status is **Status::Error**.
- All the mandatory nodes appear in error message.

**Name:**

Error when missing mandatory in a patient person

**Result:**

Success

**Description:**

The test tries to load a query file with missing mandatory values in patient person.

**Validation:**

- The import status is **Status::Error**.
- All the mandatory nodes appear in error message.

**Name:**

Error when missing mandatory in a patient institute

**Result:**

Success

**Description:**

The test tries to load a query file with missing mandatory values in patient institute.

**Validation:**

- The import status is **Status::Error**.
- All the mandatory nodes appear in error message.

**Name:**  
Retrieve complete requestXpert

**Result:**  
Success

**Description:**  
The test tries to load completely formed requestXpert.

**Validation:**

- The import status is **Status::Ok**.
- All the values are correctly returned by the getters.

**Name:**  
Retrieve default requestXpert

**Result:**  
Success

**Description:**  
The test tries to load a requestXpert that has not filled values that are optional.

**Validation:**

- The import status is **Status::Ok**.
- When using getter on the missing optional values, the corresponding default values are returned.

**Name:**  
Error when no requestXpert

**Result:**  
Success

**Description:**  
The test tries to load file that has no requestXpert.

**Validation:**

- The import status is **Status::Error**.
- The requestXpert vector got from XpertQueryData has 0 length.
- The importer error message indicates **"No requestXpert found"**.

**Name:**  
Error when missing mandatory requestXpert

**Result:**  
Success

**Description:**  
The test tries to load a requestXpert that has missing mandatory values.

**Validation:**

- The import status is **Status::Error**.
- All the mandatory nodes appear in error message.

## 7 Conclusion

<coming soon>

Euh... That would have been nice to have a conclusion to the intermediate report...



## 8 Bibliography

- Prof. THOMA Yann, prof. MPAGAMA Stellah, GUIDI Monia, 2022, TuberXpert: Clinical Decision Support System for antituberculosis medical drugs [PDF document]
- BENALLAL Nadir, 2018, Expert System for drug dosage adaptation [PDF document], Yverdon-les-Bains : Haute école d'ingénierie et de gestion du canton de Vaud. Bachelor's thesis
- CADUFF Max, 2020, Expert System for drug dosage adjustments [PDF document], Yverdon-les-Bains : Haute école d'ingénierie et de gestion du canton de Vaud. Bachelor's thesis
- BUCLIN Thierry, 2022, Les Bases de la pharmacocinétique clinique [PDF document]
- TUCUXI dev team, 2022, Tucuxi CLI Software Usability Specification [PDF document]
- TUCUXI dev team, 2022, Tucuxi Drug Model File Specification [PDF document]
- TUCUXI dev team, 2022, Tucuxi System Description [PDF document]
- DATAPHARM, 2022, EMC, 21 Feb 2022 [24 Mar 2022],  
<https://www.medicines.org.uk/emc/product/8788/smpc>

## 9 Authentication

The undersigned, Melvyn Herzig, hereby certifies that he alone conducted this work and did not use any other source than those expressly mentioned.

Yverdon, the Sunday, May 15, 2022

Melvyn Herzig

A handwritten signature in blue ink, appearing to read 'Herzig', with a long horizontal stroke extending to the right.

## 10 List of abbreviations

**ATC** *anatomical therapeutic chemical.*

**CDSS** *clinical decision support system.*

**DM** *diabetes mellitus.*

**EHR** *electronics health record.*

**HTML** *hypertext markup language, a file format for web page.*

**PDF** *portable document format, a file format.*

**PK** *pharmacokinetics.*

**TB** *tuberculosis.*

**TDM** *therapeutic drug monitoring.*

**UML** *unified modeling language, a way to make diagrams.*

**XML** *extensible markup language, a file format.*

## 11 List of figures

Figure 1 - Dosage Scheme (BUCLIN Thierry, 2022, Les Bases de la pharmacocinétique clinique [PDF document]).....	8
Figure 2 -TDM process (BUCLIN Thierry, 2022, Les Bases de la pharmacocinétique clinique [PDF document]) .....	9
Figure 3 Global application overview and components .....	11
Figure 4 Program global execution.....	17
Figure 5 Process of drug files evaluation.....	18
Figure 6 Process of dosages assessment .....	19
Figure 7 Process of samples assessment .....	20
Figure 8 Process of targets assessment.....	21
Figure 9 Decisions for adjustment request.....	22
Figure 10 Formula of the target score based on concentration C.....	29
Figure 11 Class diagram of administrative data .....	40
Figure 12 Class diagram of the requestXpert data .....	41
Figure 13 Class diagram of the TuberXpert query.....	41
Figure 14 Class diagram of the TuberXpert query importer.....	42
Figure 15 Class diagram of the TuberXpert language manager .....	43
Figure 16 Gantt chart of the initial planning .....	53
Figure 17 Gantt chart of the intermediate planning .....	54

## 12 Planification

This chapter presents the Gantt charts of the planning. The initial diagram was made after a couple of hours of work, the intermediate diagram after 150 hours and the final diagram after 450 hours.

### 12.1 Initial

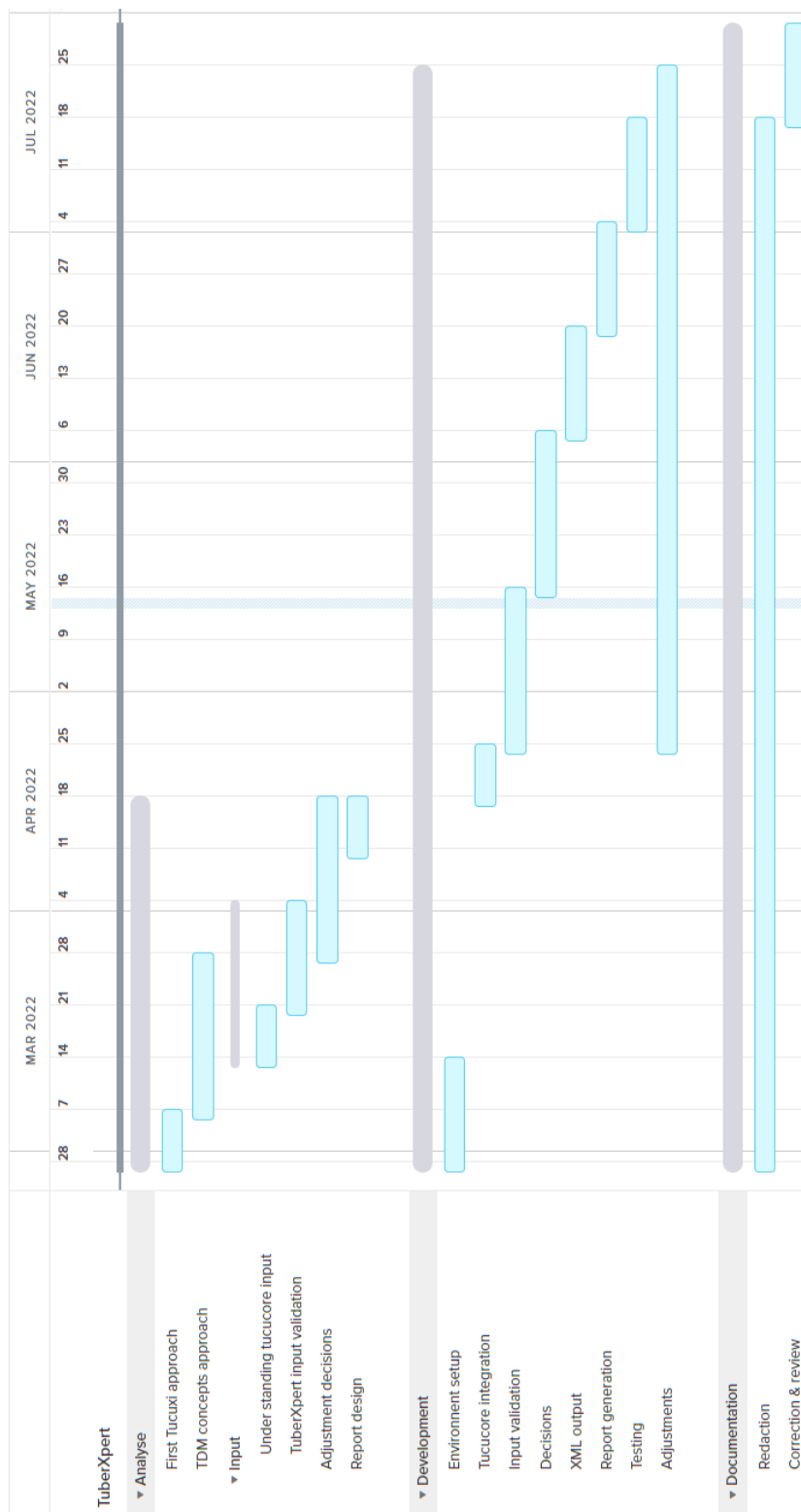


Figure 16 Gantt chart of the initial planning

## 12.2 Intermediate

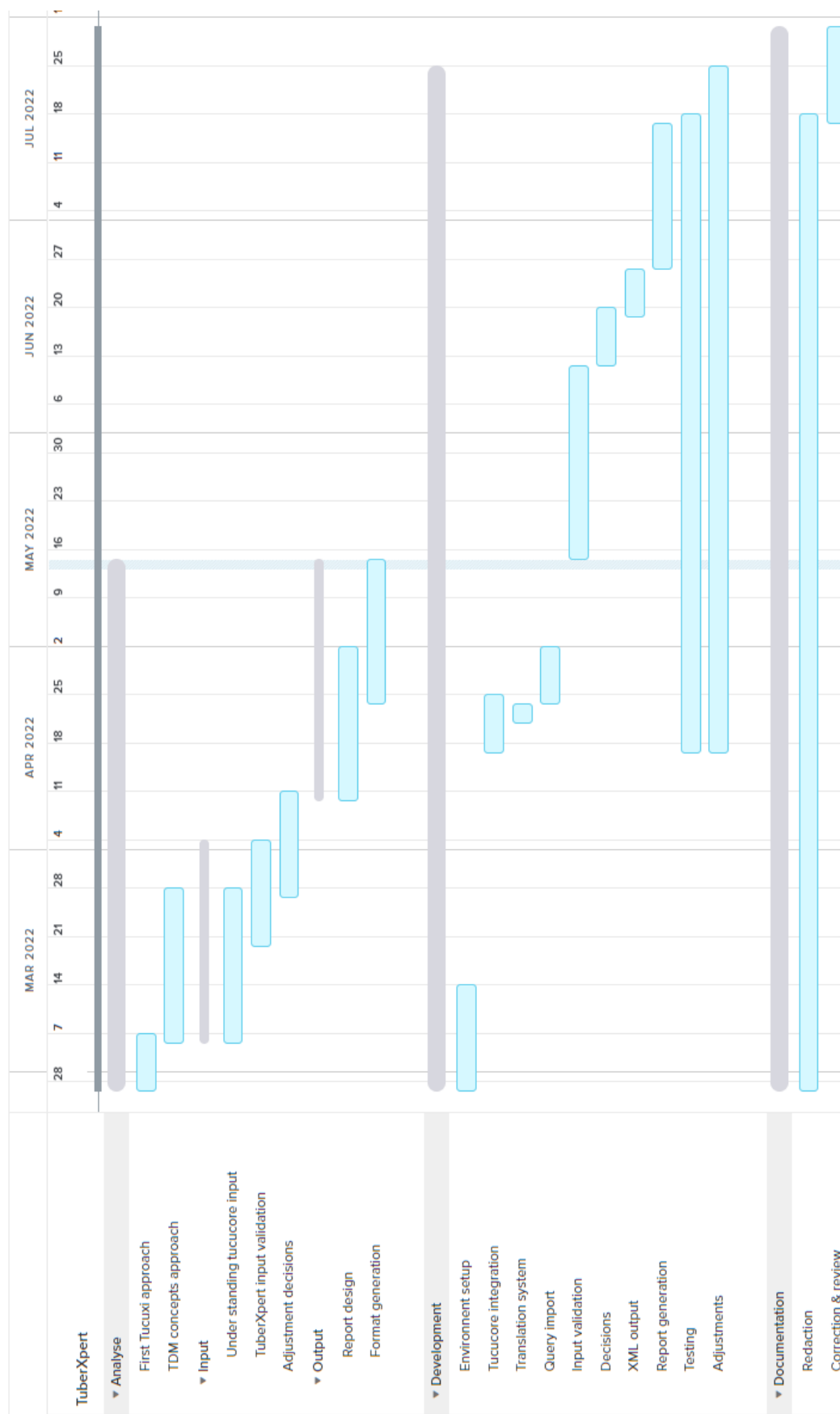


Figure 17 Gantt chart of the intermediate planning

## 12.3 Final

<coming soon>